ALIEN TECHNOLOGY®

READER INTERFACE GUIDE

All Fixed Readers

July 2014

ALR-9900+
ALR-9900
ALR-9680
ALR-9650

Legal Notices

Alien Technology Corporation has intellectual property rights relating to technology embodied in the products described in this document, including without limitation certain patents or patent pending applications in the U.S. or other countries.

This Product includes certain open source software ("Program") licensed pursuant to the GNU General Public License, Version 2, 1991 (the "License"). Pursuant to the terms of the License, we will make available to you, for three years from the date of purchase of this Product from Alien Technology Corporation, a complete machine-readable copy of the source code of the Program. We will provide the Program to you on a medium customarily used for software interchange, and may charge you an amount no more than our cost of physically performing source distribution.

This product is covered by one or more of the following U.S. patents: 7716208, 7716160, 7688206, 7671720, 7659822, 7619531, 7615479, 7598867, 7580378, 7576656, 7562083, 7561221, 7559486, 7559131, 7554451, 7411503, 7385284, 7377445, 7364084, 7353598, 7342490, 7324061, 7321159, 7301458, 7295114, 7288432, 7265675, 7262686, 7215249, 7214569, 7199527, 7193504, 7173528, 7172910, 7172789, 7141176, 7113250, 7101502, 7080444, 7070851, 7068224, 7046328, 6998644, 6988667, 6985361, 6980184, 6970219, 6952157. Other patents pending.

DOCUMENTATION IS PROVIDED “AS IS” AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARANTEES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGMENT ARE HEREBY DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

European Radio Compliance

This Equipment has been tested and found compliant with the following Mandatory Specifications:

EN 302 208
Electromagnetic Compatibility and Radio spectrum Matters (ERM)
Radio Frequency Identification Equipment operating in the band 865 MHz to 868 MHz with Power levels of 2w ERP.
Part 1: Technical requirements and methods of measurements.
Part 2: Harmonized EN under article 3.2 of R&TTE Directive.

EN 300 220
(when so marked on the equipment label)
Electromagnetic Compatibility and Radio spectrum Matters (ERM):
Short Range Devices (SRD);
Radio equipment to be used in the 25 MHz to 1000 MHz frequency range with power levels ranging up to 500 mW;
Part 1: Technical characteristics and test methods
Part 2: Harmonized EN covering essential requirements under article 3.2 of the R&TTE Directive

Note: It is the user’s responsibility to operate the device in compliance with the 10% duty cycle restriction as described in this manual.

EN 301 489
Electromagnetic compatibility and Radio spectrum Matters (ERM)
Electromagnetic Compatibility (EMC) standards for Radio equipment and services:
Part 1: Common technical requirements
Part 3: Specific conditions for short-Range Devices (SRD) operating on frequencies between 9 kHz and 40 GHz.

EN60950
Safety of information technology equipment.
This radio must be used with the external mains to DC power adaptor, supplied with the product.

CE 0891 !
CE 0681 !

ALIEN

Alien Technology®

Reader Interface Guide

ALR-9900/9900+, ALR-9680, ALR-9650

July, 2014

Table of Contents

CHAPTER 1
Introduction and Reader Setup

This *Reader Interface Guide* provides instructions for installing and operating the following Alien Technology® RFID readers:

- ALR-9900, ALR-9900+ (Enterprise)
  Note: ALR-9900+ supports the same commands as the ALR-9900 with a few exceptions explicitly specified in this manual.
- ALR-9680
- ALR-9650 (Smart Antenna)

This guide also details the protocol used between a host and these readers for system configuration and the acquisition of data by application software.

This document is designed for use by RFID system integrators and software developers - those who wish to develop software products and extended systems that take full advantage of the RFID Reader's capabilities.

For an overview of RFID technology and a glossary of terms, please refer to the Alien *RFID Primer* document.

Audience

For the purposes of this document, we assume the readers of this *Reader Interface Guide*:

- are competent PC users
- may be IT specialists, network specialists or programmers
- have minimal previous knowledge of RFID technology
- are experienced in software development and/or hardware systems integration

Additionally, it is assumed that:

- Users installing the reader via direct serial communication are skilled in the application of RS-232 serial protocol.
- Users installing the reader for network communication are skilled in basic network configuration.
- Programmers are competent in at least one programming or scripting language and have the ability to issue ASCII-based commands with that language.

Type Conventions

- Regular text appears in a plain, sans-serif font.
- External files and documents are referenced in *italic text*.
- Specific characters and commands to be typed are shown within quotation marks, and/or in `fixed-width` font. Example: At the prompt type "`DHCP=ON`".

- Values to be provided and typed in by the user are shown within brackets in upper and lowercase. Example: At the prompt type "`IPAddress=[IP address value]`" or "`IPAddress=xxx.xxx.xxx.xxx`". The actual command typed in would appear as: "`IPAddress=10.1.60.5`".
- Blocks of sample code or commands appear:

```
indented, in a fixed-width serif font.
```

- Keys to be pressed are shown in brackets and all caps. Example: Press the [ENTER] key.
- Upon entering any command instruction, you must press [ENTER] to send the command.
- RFID Reader commands are not case sensitive. Although, for clarity, the commands may be shown in upper and lower case in this document, you may type them in all lowercase characters, if you prefer.
- A space is required between the command (verb) such as "get" or "set" and the specific parameters, as in the example "`get IPAddress`". However, no space is required between the parameter elements such as "IP" and "address."

Requirements

In order to fully interface with the RFID Reader you need the following:

- a PC running Windows and an available RS-232 serial port
- host software (Alien RFID Gateway demo software, or your own custom software)
- RFID Tags (AIDC Class I compliant)

Serial communication requires:

- a serial communications program (such as HyperTerminal, TeraTerm, or PuTTy) running on any computer

Ethernet communication requires:

- an Ethernet network
- a Telnet communication program

Communicating with the Reader

This section of the *Reader Interface Guide* describes how to connect the reader on a host computer, as well as how to issue commands and interact with the reader using two different communication methods: serial (RS-232) and Telnet (TCP/IP).

Whether using direct serial communication with the reader or using one of the network communication options, you may still need serial communications for initial reader setup.

Alien also provides extensive examples of code that use Java, Visual Basic, and .NET programming languages. These examples serve as models for developing new software for the reader, and can be found on the reader CD (if provided with your kit), at the Alien Partner Portal website (http://partners.alientechnology.com), or on the public FTP server (ftp://ftp.alientechnology.com).

Serial Communication

This method is helpful for installing a new RFID Reader. Serial communication requires no pre-configuration and can be performed easily with most computers. This method enables real-time operation of the reader via a serial communications (COM) port. Serial communication is the simplest means by which to interface with the reader and implement the Alien Reader Protocol.

The reader is configured to use DHCP to acquire its network settings, and while this method of network configuration is simple and convenient, the problem exists that in order to communicate with the reader, you have to know its IP address. Alien readers have a heartbeat mechanism to assist in discovery of readers on the network, but this mechanism requires a host application (such as the Gateway demonstration software) to intercept the heartbeat messages and report them back to you. In these circumstances, the serial interface can be used to determine the reader's network address.

Serial Configuration

Whether you will ultimately be operating the reader directly via serial communications or via a network connection, you need to install the reader initially using the serial port instructions.

NOTE: Example screens shown in this section are from HyperTerminal.

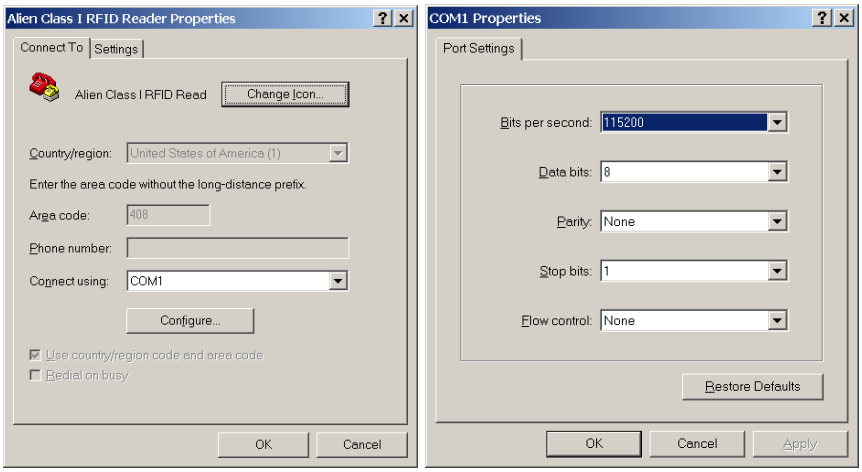
1. Ensure the reader is properly connected to power and at least one antenna (ALR-x800 readers require two antennas). See your reader's *Hardware Setup Guide* for more information.

2. Connect one end of the serial cable to the reader's RS-232 port and the other end to an available COM port on the host computer.

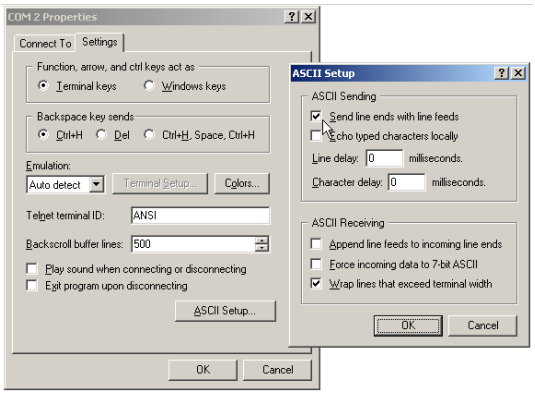
3. Launch the desired serial communications program, such as HyperTerminal, TeraTerm, or PuTTy.

4. Enter (or verify) the following settings to configure the serial communications program:
   Baud Rate : 115200
   Data Bits : 8
   Parity : None
   Stop Bits : 1
   Flow Control : None

Once configured, the software should allow you to communicate with the RFID Reader. HyperTerminal example configuration screens are shown below:



5. It is also a good idea to make sure your terminal software sends linefeeds as well as carriage returns.



6. Power up the reader, and observe the information reported over the serial port. Perhaps the most important bits of information are the network settings, since you need to know this in order to communicate with the reader using TCP/IP.

The reader displays a block of text similar to the following, toward the end of the bootup:

```
==========================================
------------------------------------------
Network Settings:
    MAC Address : 00:80:66:10:2D:12
    Hostname    : alien-102D12
    DHCP        : 1
    DHCPTimeout : 30
    IP Address  : 10.9.8.10
    Netmask     : 255.255.255.0
    Gateway     : 10.9.8.2
    DNS         : 10.9.8.1
    TimeServer  : time-a.timefreq.bldrdoc.gov
    TimeZone    : -7
    CommandPort : 23
------------------------------------------
==========================================
```

This trace indicates that the reader can be contacted over TCP/IP at the address 10.9.8.10, on port 23.

7. You eventually see the “Boot> Ready” indicator, and after pressing the [ENTER] key you see the "Alien>" command prompt. At the command prompt, you may now type any reader command followed by the [ENTER] key to submit the command.

The following basic commands are helpful in verifying the reader-host interface:

- "Help" (or “h”) – provides a list of all commands available
- "Info" (or “i”) – provides a list of current settings for the reader
- "get TagList" (or "t") – scans field immediately for tags and reports the results

NOTE: RFID Reader commands are not case-sensitive and may be typed in all lowercase characters, if preferred.

For a detailed explanation of all commands available, please refer to the chapter titled Alien Reader Protocol.

Network Communication

TCP/IP communication requires a network connection via the reader's Ethernet port and allows the reader to operate like a Telnet server.

This mode offers the same form of command line interaction with the RFID Reader as the serial interface, but requires the RFID Reader to be configured for and running on a network in order to use it.

By default, all RFID Readers are pre-configured to use DHCP when presented with an Ethernet connection. However, you must first establish a direct serial connection in order to learn the reader's IP Address, or be able to detect "heartbeat" messages sent out over the network by the reader.

Determine the Reader's Network Settings

Once your readers are set up and configured, you should either keep a record of their addresses, or use some other mechanism for finding them, such as the reader's Heartbeat messages (described later). Following are instructions for querying the reader for this information.

There are five network commands that are used for querying the reader for its network configuration. These commands can be issued directly in the reader's serial interface.

- DHCP?
- IPAddress?
- Netmask?
- Gateway?
- DNS?

Another alternative is to issue the "info network" command to get a summary of network settings.
Once you have the IPAddress and Netmask of the reader, you can connect to it using TCP/IP.

Connecting via TCP/IP

The reader's TCP/IP interface mimics a basic telnet interface, so you can use any telnet client to connect to the reader. Windows users can use the telnet command at the command prompt or "Start -> Run" window:

```
telnet <IPAddress> <CommandPort>
```

For instance, in the earlier example, the command would be:

```
telnet 10.9.8.10 23
```

To do this programmatically, simply open a TCP socket to the reader on its command port (default is 23) and begin reading. Refer to chapter 4, *Alien Reader Protocol*, for details on how to properly terminate command strings, detect the end of a reader's response, and disable the "Alien>" prompt text from the returned data. Default settings are:

```
Username = alien
Password = password
```

You are now ready to interact with the reader. For telnet operation, you use the same text-based commands as in direct serial communication. The only difference is in the use of the “q” command to quit the telnet session and disconnect the session. The reader automatically disconnects an idle TCP/IP connection, if there has been no activity longer than the NetworkTimeout setting.

Reader commands and instructions on their use are provided later in the chapter titled, *Alien Reader Protocol*.

TCP/IP Configuration

To configure the system for network operation, you use the commands shown under the "NETWORK" heading of the reader's "Help" display.

There are five network commands that are used for network configuration:

- DHCP
- IPAddress
- Netmask
- Gateway
- DNS

1. To view the command list, type “h” or “help”.

If DHCP is supported at your site:

2. Type “`DHCP=ON`”. DHCP automatically configures the rest of the network parameters the next time the reader boots.

3. Skip to step 7.

If DHCP is <u>not</u> supported at your site:

4. Type “`DHCP=OFF`”. The reader returns the message “DHCP = OFF”.

5. Contact your system administrator for the following parameter values:
   IPAddress
   Netmask (also known as Subnet Mask)
   Gateway
   DNS (optional)

6. Type each of the 4 commands below with the assigned values:

```
IPAddress = xxx.xxx.xxx.xxx
Netmask = xxx.xxx.xxx.xxx
Gateway = xxx.xxx.xxx.xxx
DNS = xxx.xxx.xxx.xxx (optional)
```

   As each value is accepted, the reader replies with the accepted value.

   Setting incorrect values for Gateway and DNS may result in the reader taking longer to boot up. This is caused by failed attempts to resolve various internet addresses. An invalid Gateway address may prevent the reader from initiating its own TCP network connections (tag streaming, automode notifications, etc.)

7. Type “`Save`” to direct the reader to store the new values in flash memory, then “`Reboot`” to reboot the reader and apply the new network settings.

CHAPTER 2
Reader Fundamentals

This chapter provides an overview of the major features found in an Alien RFID Reader. Specific instructions for setting up a reader are provided in the previous chapter. Reader commands and their uses are covered in the next chapter.

Introduction

The most basic function of the RFID Reader is to read RFID tags and to give a user or application access to a list of these tags. The RFID Reader is designed to perform this function either connected to a host via serial cable, or on a network.

To assist in the administration of networked operation, the reader has two important features designed to simplify network management:

- Reader "heartbeats" allow network applications to easily discover readers on a network.
- AutoMode allows unattended readers to look for tags and send notification messages to listening services on the network when certain conditions arise.

These important concepts, along with the basics of communicating with the reader, are discussed in this chapter.

Reader Discovery and the Reader Heartbeat

One of the problems common to many network appliances is simply discovering the address of the device on the network. To operate these devices over the network, users must know the device's IP address.

If an IP address is hard-coded into the device, this problem is solved, and often a label on the device can be used to indicate the IP address.

However, many systems do not use hard-coded IP addresses, requiring the network to assign an address each time the device is booted. This is called DHCP, which stands for Dynamic Host Configuration Protocol.

DHCP and Automatic Discovery

The DHCP mode of configuration eliminates the need for the user to perform network configuration for the device. The device simply is plugged into the network socket, booted, and immediately becomes a citizen of the network, acquiring its network settings directly from the DHCP server.

However, the user still needs to learn the IP address of the device. All that is known at this point is that the device does have an IP address and has booted itself on the network. The actual IP address the device is using is still not known.

Serial Interrogation

One of the simplest methods to find out the reader's IP address is to connect via the RS-232 port and type the command "`IPAddress?`". The reader responds with the IP address currently in use.

However, this requires a physical connection between a host computer and the reader —a connection that in many cases is simply not practical to set up.

Network Heartbeats

Another way to find out a reader's IP address is to listen for its heartbeat messages over the network.

After a reader has booted successfully onto a network it periodically broadcasts a heartbeat message over the network. This heartbeat can be intercepted by network applications, and provides enough information about the reader to locate it on the network and begin communication with it.

In network parlance, the heartbeat message is sent via UDP (Universal Datagram Protocol) packets to all network addresses on the reader's subnet.

There are three relevant configuration options available via the reader's command line to affect this heartbeat:

- **HeartbeatTime:** This command specifies the time interval separating successive heartbeat messages sent out over the network. The time is specified in seconds, with a value of zero turning off the heartbeats. The default value is 30 seconds, i.e. send out a heartbeat message every 30 seconds.
- **HeartbeatPort.** This command specifies the port number that the UDP heartbeat messages are addressed to. This is the port number that must be listened to by interested parties on the network. The default value for this setting is 3988, i.e. send out a heartbeat message to UDP port 3988 of every machine on the subnet.
- **HeartbeatAddress.** This command specifies a particular IP Address that the UDP heartbeat messages are addressed to. The default value is 255.255.255.255, which is a special "multicast" addressed which allows any machine on the subnet to receive the heartbeats. Any other HeartbeatAddress results in the heartbeats going only to the machine at that address.
- **HeartbeatCount.** This command allows you to specify the total number of heartbeat messages to send. This allows for the immediate discovery of readers as they come online, but doesn't continuously load the network with unnecessary traffic.

The format of the heartbeat is a small XML text-based message, containing information about the reader (name and type), the reader's network connection (IP address and command port) and the length of time until the next heartbeat is sent out.

```
<Alien-RFID-Reader-Heartbeat>
   <ReaderName>Alien RFID Reader</ReaderName>
   <ReaderType>
      Alien RFID Tag Reader, Model: ALR-9900
      (Four Antenna / Gen 2 / 902-928 MHz)
   </ReaderType>
   <IPAddress>10.1.60.5</IPAddress>
   <CommandPort>23</CommandPort>
   <HeartbeatTime>30</HeartbeatTime>
   <MACAddress>01:02:03:04:05:06</MACAddress>
   <ReaderVersion>07.07.18.00</ReaderVersion>
</Alien-RFID-Reader-Heartbeat>
```

HEARTBEAT XML TAGS

- **ReaderName** is the user-defined name associated with the reader. This name can be set by a user to help identify which reader is which. For example, multiple readers in a warehouse may be named "loading bay 1", "loading bay 2" etc., thus providing a clear indication as to the physical location of the reader.

- **ReaderType** details the specific type of reader sending out the heartbeat. This information is hard-coded into the reader's firmware and is not user-configurable.
- **IPAddress and CommandPort** elements detail the location of the reader on the network. The IP address is simply the network address of the reader. The command port is the port number on which the reader is listening for incoming user commands. Typically, this is port 23, the standard telnet port, allowing a user to communicate with the reader over the network by typing "telnet [IPAddress]" into the command line of most computers.
- **HeartbeatTime** is the time until the next heartbeat. This time (in seconds) enables any application software to detect when a reader is powered-down or the network connection breaks; if a new heartbeat is not received after the expected time elapses, then such an interruption to normal service can be detected.
- **MACAddress** gives the unique identifier assigned to the reader's network interface hardware by the manufacturer.
- **ReaderVersion** gives the version information for the software currently running in the reader.

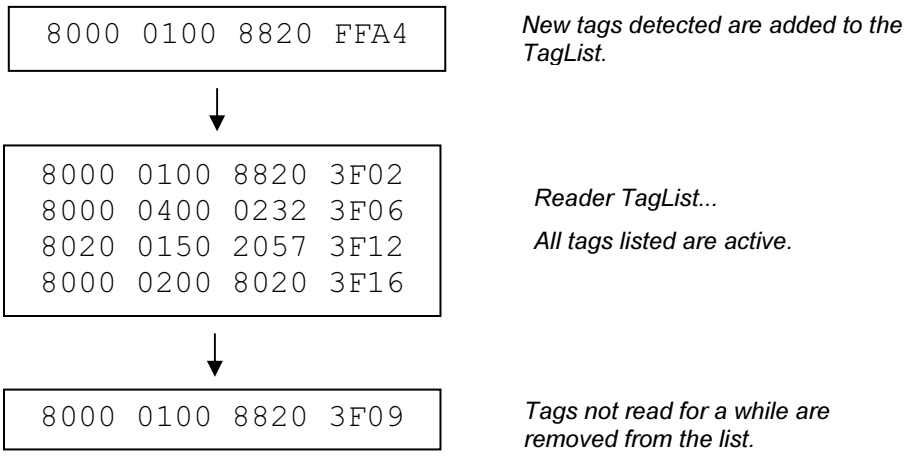
HEARTBEATS AND SOFTWARE

The Alien SDK provides source code and software libraries in Java, .NET, and Visual Basic to listen for these network heartbeats.

The Alien RFID Gateway application uses the Java version of these libraries to build its active reader list on the main screen. The latest Gateway version at the time of this writing is v2.25.00. Alien recommends users first download and install the latest version of Gateway software.

TagList Concepts

During normal operation, the reader always has a concept of "what's out there" by maintaining an internal list of the tags that are *active*. Active tags are those read by the reader at least once within a predefined interval. Any new tags presented to the reader are added to the list, and any tags that have not been seen for a period of time (the PersistTime) are removed from the list.



At any time, a programmatic call can be made to the reader to retrieve this list of tags.

PersistTime
The “persist time” defines the duration between the time a tag was last read and the time it is automatically purged from the TagList. Setting this value to a small time (~1 second) causes the TagList to contain only what the reader has seen in the last second, i.e., a fair representation of what the reader sees at any one time. Setting the PersistTime to a long duration allows a history of tags to be built up. For example, setting the PersistTime to 3600 seconds allows a list to be built up detailing all the tags that were read over the last hour.

Setting PersistTime to -1 has the special effect of keeping tags on the list indefinitely until the TagList is delivered to a host, at which point the TagList is cleared. This is the default value.

Tag Details
Each entry in the TagList contains a number of sub fields: the Tag's unique ID, the read count (the number of times the tag has been read in the current session), the discovery time (the time the tag was first seen), the last observed time, the antenna (the antenna ID that the tag was last read from), and others.

TagList Size
The TagList can hold up to 6000 tag entries. Some reader models also have the ability to retain some of the TagList data across power outages,

Reading Tags over the Network

The Alien RFID reader provides two methods with which to read tags: Interactive Mode and Autonomous Mode.

- **Interactive Mode** - the controlling application issues commands to the reader to read tags. This command always immediately returns with a list of tags in the reader's field of view.
- **Autonomous Mode -** the reader constantly reads tags, and may initiate a conversation with a network listener when certain events arise.

While both methods are equally useful, the choice ultimately is determined by the needs of the controlling application.

Although it may be easier and require less coding to work in Interactive Mode, a little investment in programming effort lets the user set up AutoMode to provide a more scalable system for multiple readers.

Interactive Mode

Reading tags in Interactive Mode is as simple as issuing a single command to the reader. This command is “`TagList?`“, or simply "t". This causes the reader to initiate a tag search, and report back the current TagList. Depending on the PersistTime and recent tag activity, the returned data may include tag read data from previous reads. Following is an example of what the default TagList format ("Text") looks like.

```
Tag:E200 3411 B801 0108, Disc:2007/06/29 08:30:49, Last:2007/06/29 10:38:12, Count:292, Ant:0, Proto:2
Tag:4461 7669 6445 2E4B, Disc:2007/06/29 10:38:13, Last:2007/06/29 10:38:13, Count:187, Ant:1, Proto:2
```

The format of the TagList can be specified using the “`TagListFormat`“ command. One of the options is XML format, which would return the same TagList as:

```
<?xml version="1.0" encoding="UTF-8"?>
<Alien-RFID-Tag-List>
   <Alien-RFID-Tag>
      <TagID> E200 3411 B801 0108</TagID>
      <DiscoveryTime>2007/06/29 08:30:49</DiscoveryTime>
      <LastSeenTime>2007/06/29 10:38:12</LastSeenTime>
      <ReadCount>292</ReadCount>
      <Antenna>0</Antenna>
      <Protocol>2</Protocol>
   </Alien-RFID-Tag>
   <Alien-RFID-Tag>
      <TagID>1155 4461 7669 6445 2E4B</TagID>
      <DiscoveryTime>2007/06/29 10:38:13</DiscoveryTime>
      <LastSeenTime>2007/06/29 10:38:13</LastSeenTime>
      <ReadCount>187</ReadCount>
      <Antenna>1</Antenna>
      <Protocol>2</Protocol>
   </Alien-RFID-Tag>
</Alien-RFID-Tag-List>
```

Additional formats include "Terse", and "Custom". See the TagListFormat section in Chapter 4, *Alien Reader Protocol* for details on all of the TagList formats.

Autonomous Mode

AutoMode is a configuration and operation mode that enables automated monitoring and handling of tag data. You first issue a series of configuration commands to the reader which specify how and when to read tags, and optionally what to do with the tag data.

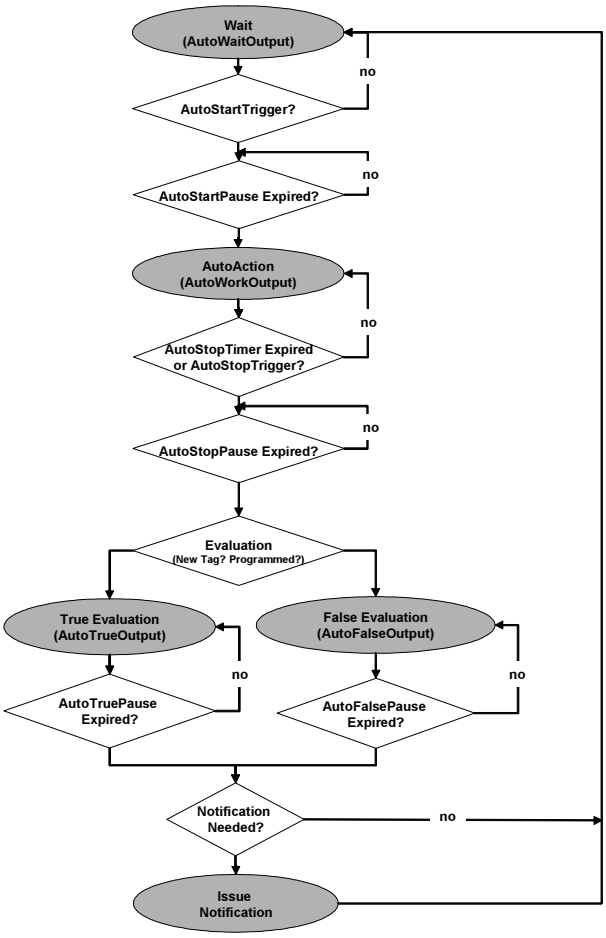
Once configured in this mode, the reader can be left to operate on its own. An application on a host computer can then be set up to listen for notification messages from the reader containing any tag data that it has read.

One of the major benefits to this mode of operation is that many readers can be configured to send tag messages to a single host. Thus, a single application can listen for and process data from multiple readers over the network.

Defining the Autonomous Read Operation

AutoMode functionality is summarized in the state diagram shown below. Fundamentally, a reader operating in AutoMode moves between several states: Waiting, Working, Evaluation, and Notification. Waiting, Working, and Evaluation states have associated with them an optional digital output state that is set upon entering the state. Movement from one state to the next is initiated by an expiration of a timer, a triggered event on the digital input lines, or changes to the TagList.

Each element of the State Diagram is described below. Associated with each element are one or more commands that are used to configure the reader.



Autonomous Mode State Diagram

ENTER AUTOMODE

(Not shown on the state diagram.)

The user puts the reader into AutoMode with the "`AutoMode`" command. "`AutoMode=On`" puts the reader into AutoMode. "`AutoMode=Off`" turns off AutoMode.

WAITING STATE

Upon entering AutoMode, the reader automatically enters the Waiting State. While waiting for a Start Working Trigger (see below) the reader holds the digital output lines at a value set by the `AutoWaitOutput` command. For example, "`AutoWaitOutput=1`" causes digital output line 1 to go high while the reader is in the Waiting state.

START WORKING TRIGGER

The receipt of a trigger pattern on the digital input lines causes the reader to move from the Waiting state to the Working state. The start condition is set by the `AutoStartTrigger` command. The `AutoStartTrigger` command takes two parameters, a rising edge pattern and a falling edge pattern. Each pattern is a single integer which is a bitmap of the desired input triggers, where input #1 is represented by bit 0, input #2 is represented by bit 1, etc.

The table below illustrates the bitmap values that correspond to different I/O combinations. This table holds for both input as well as output bitmaps.

Digital I/O Value	I/O #4	I/O #3	I/O #2	I/O #1
0	-	-	-	-
1	-	-	-	✓
2	-	-	✓	
3	-	-	✓	✓
4	-	✓	-	-
5	-	✓	-	✓
6	-	✓	✓	-
7	-	✓	✓	✓
8	✓	-	-	-
9	✓	-	-	✓
10	✓	-	✓	-
11	✓	-	✓	✓
12	✓	✓	-	-
13	✓	✓	-	✓
14	✓	✓	✓	-
15	✓	✓	✓	✓

The command, "`AutoStartTrigger=2 0`", causes the reader to enter the working state on receipt of a rising edge on input #2, while "`AutoStartTrigger=0 3`" causes the reader to enter the working state after the receipt of a falling edge on input #1 or input #2. "`AutoStartTrigger=0 0`" causes the reader to immediately drop into the Working state.

The AutoStartPause property causes the reader to wait the specified number of milliseconds after receiving a start trigger before transitioning to the working state.

WORKING STATE

In the working state, the reader holds the digital output lines at the value defined by the `AutoWorkOutput` command. "`AutoWorkOutput=3`" holds the first two output lines high while the reader is working. The action the reader performs while in the working state is determined by the `AutoAction` command. "`AutoAction=Acquire`" causes the reader to repeatedly acquire TagList data using the parameters set in the `AcquireMode` and `PersistTime` commands. The reader continues working until the Stop Working Trigger conditions are met (see below).

STOP WORKING TRIGGER/TIMER

Like the Start Working Trigger, the Stop Working Trigger can be a change on the digital input lines, but can also be the expiration of a timer. Use the `AutoStopTrigger` command with a rising, falling edge pattern to set the trigger conditions. "`AutoStopTrigger=1 0`" looks for a rising edge on input #1 before leaving the Working state. In addition, one may use the `AutoStopTimer` command to repeat the Working action for a specified period of time. For example, "`AutoStopTimer=1300`" causes the reader to perform the Working action for 1.3 seconds before moving on to the Evaluation stage. If both a stop trigger and a stop timer are specified, whichever event occurs first stops the working state.

EVALUATION

At the Evaluation decision point, the reader looks to see if new tags have been added to the TagList since the last evaluation. If so, it drops to the AutoTruePause state, if not, it drops to the AutoFalsePause state.

Note: the Evaluation looks at the TagList and therefore is very much dependent on the value of the PersistTime setting.

TRUE/FALSE PAUSE

After evaluation, the Reader sets the output lines to the values specified in the `AutoTrueOutput` and `AutoFalseOutput` commands. This condition is held for `AutoTruePause` or `AutoFalsePause` milliseconds before the test for Notification begins. For example, "`AutoTrueOutput=1`" and "`AutoTruePause=20`" cause the reader to hold output #1 high (and all others low) for 20 milliseconds before proceeding.

NOTIFICATION

If NotifyMode is enabled (`NotifyMode=On`) and the specified NotifyTrigger has occurred, the reader issues an notification message. The NotifyTrigger is specified with the "`NotifyTrigger`" command and can be set to "Add", "Remove", "Change", "True", "False", or "TrueFalse", as described in the next topic.

If a notification is to be issued, the TagList data is sent to the `NotifyAddress`. The reader then returns to the Waiting state.

AutoMode Examples

EXAMPLE 1 - BACKGROUND READING

In this case, we would like the reader to monitor the tag field continuously. The application periodically asks for the TagList. If a new tag is seen, output #1 flashes high for 500 msec. Otherwise, output #2 flashes high for 500 msec.

```
AutoModeReset
AutoAction = Acquire
AutoStartTrigger = 0,0
AutoStopTimer = 0
AutoTrueOutput = 1
AutoTruePause = 500
AutoFalseOutput = 2
AutoFalsePause = 500
AutoMode = On
```

EXAMPLE 2 - TRIGGERED READING

Here a forklift causes an electric eye to send a signal to the reader. We want the reader to look for a rising edge on this signal and scan for tags for 1.8 seconds before going back to the Waiting state. We won't make any changes to the outputs.

```
AutoModeReset
AutoAction = Acquire
AutoStartTrigger = 1, 0
AutoStopTimer = 1800
AutoTruePause = 0
AutoFalsePause = 0
AutoMode = On
```

EXAMPLE 3 - TRIGGERED READING WITH NOTIFICATION

A trigger is used to start the reading. If a tag is found, send an email message. After the email is sent, return to the waiting state.

```
AutoModeReset
AutoAction = Acquire
AutoStartTrigger = 1, 0
AutoStopTimer = 0
AutoTruePause = 0
AutoFalsePause = 0
NotifyAddress = borg@mycompany.com
MailServer = mail.mycompany.com
NotifyTrigger = Add
NotifyMode = On
AutoMode = On
```

Notification Mode

The last stage in configuring AutoMode is to tell the reader under what conditions to notify listeners about TagLists. Listeners (network applications / people) are notified only when specific conditions arise, such as when new tags are read, or when tags disappear from view.

NotifyTime

The NotifyTime command instructs the reader to send out a copy of its TagList to a listener every n seconds, regardless of whether the TagList has changed or not. This is a simple way to force the reader to send out its TagList to a listener.

NotifyTrigger

The NotifyTrigger command specifies a condition that must occur before a TagList is sent out to a listener. There are a number of possible triggers that can be used:

Trigger	Trigger Condition	Tag Data Included
Add	A new tag was read and added to the TagList.	Only the added tags.
Remove	A tag was removed from the TagList.	Only the removed tags.
AddRemove	A tag was added to or removed from the TagList	A list of added tags, then a list or removed tags.
Change	A tag was either added to, or removed from, the TagList.	Entire TagList.
True	The evaluation task of the autonomous state loop evaluates to `true` (typically when a tag is added to the TagList).	Entire TagList.
False	The evaluation task of the autonomous state loop evaluates to `false` (typically when no tag is added to the TagList)	Entire TagList.
TrueFalse	The evaluation task of the autonomous state loop evaluates to `true` or `false` (i.e. every AutoMode cycle)	Entire TagList.

NotifyAddress

The NotifyAddress is the location where the reader delivers notification messages to. The reader can be instructed to send messages to a specific machine on the network over a TCP socket, via email to a specific email address, or even out the serial port. This is configured using the command:

```
NotifyAddress = <address>
```

The format of <address> indicates the method of delivery:

NotifyAddress	Description
hostname:port	Send a message to a specified port on a networked machine. The address takes the form “hostname:port.” For example, “123.01.02.98:3450” or "listener.alientechnology.com:10002”
user@domain.com	Send a message via e-mail to the address specified. The address is specified in standard email form, i.e., user@domain.com *NOTE: the MailServer parameter must be configured for this to work. No email authentication protocols are supported at this time.*
SERIAL	Send a message to the serial connection. The word “SERIAL” is used as the address, and is not case sensitive.

NotifyFormat

You can tell the reader the format for any notification that it issues. When a notification message is sent out, it contains two parts:

- The first part, the header, provides details about the reader that sent the message, and the reason the message was sent.
- The second part is the TagList - either newly added or removed tags, or the complete list of tags as seen by the reader, depending on the NotifyTrigger.

The format of the message is configured using a single command:

```
NotifyFormat = <format>
```

The format may be one of the following:

NotifyFormat	Description
Text	Plain text messages, one tag ID per line.
Terse	Plain text messages, one tag ID per line, compact form
XML	XML text format
Custom	Same as Text format, except the contents of each tag ID line is defined by the TaglistCustomFormat parameter

Text-formatted notifications take the form:

```
#Alien RFID Reader Auto Notification Message
#ReaderName: Spinner Reader
#ReaderType: Alien RFID Tag Reader, Model: ALR-9900
             (Four Antenna / Gen 2 / 902-928 MHz)
#IPAddress: 10.1.70.13
#CommandPort: 23
#MACAddress: 00:80:66:10:11:6A
#Time: 2003/01/21 12:48:59
#Reason: TEST MESSAGE
#StartTriggerLines: 0
#StopTriggerLines: 0
Tag:1115 F268 81C3 C012, Disc:2003/01/21 09:00:51, Last:2003/01/21 09:00:51, Count:1,
Ant:0, Proto:1
Tag:0100 0100 0002 0709,  Disc:2003/01/21 11:00:10, Last:2003/01/21 11:00:10, Count:1,
Ant:0, Proto:1
#End of Notification Message
```

Terse-formatted notifications take the form:

```
#Alien RFID Reader Auto Notification Message
#ReaderName: Spinner Reader
#ReaderType: Alien RFID Tag Reader, Model: ALR-9900
             (Four Antenna / Gen 2 / 902-928 MHz)
#IPAddress: 10.1.70.13
#CommandPort: 23
#MACAddress: 00:80:66:10:11:6A
#Time: 2003/01/21 12:48:59
#Reason: TEST MESSAGE
#StartTriggerLines: 0
#StopTriggerLines: 0
1115 F268 81C3 C012,0,1
0100 0100 0002 0709,0,1
#End of Notification Message
```

XML-formatted notifications take the form:

```
<?xml version="1.0" encoding="UTF-8"?>
<Alien-RFID-Reader-Auto-Notification>
    <ReaderName>Spinner Reader</ReaderName>
    <ReaderType>
        Alien RFID Tag Reader, Model: ALR-9900 (Four Antenna / Gen 2 / 902-928 MHz)
    </ReaderType>
    <IPAddress>10.1.70.13</IPAddress>
    <CommandPort>23</CommandPort>
  <MACAddress>00:80:66:10:11:6A</MACAddress>
    <Time>2003/01/21 12:49:22</Time>
    <Reason>TEST MESSAGE</Reason>
    <Alien-RFID-Tag-List>
        <Alien-RFID-Tag>
          <TagID>0102 0304 0506 0709</TagID>
          <DiscoveryTime>2003/01/17 11:37:01</DiscoveryTime>
          <LastSeenTime>2003/01/17 11:37:01</LastSeenTime>
          <Antenna>0</Antenna>
          <ReadCount>1413726</ReadCount>
          <Protocol>1</Protocol>
        </Alien-RFID-Tag>
        <Alien-RFID-Tag>
        <TagID>2283 1668 ADC3 E804</TagID>
        <DiscoveryTime>2003/01/19 07:01:19</DiscoveryTime>
        <LastSeenTime>2003/01/19 07:01:19</LastSeenTime>
        <Antenna>0</Antenna>
        <ReadCount>1</ReadCount>
          <Protocol>1</Protocol>
        </Alien-RFID-Tag>
    </Alien-RFID-Tag-List>
</Alien-RFID-Reader-Auto-Notification>
```

An example of a custom-formatted notification might be:

```
#Alien RFID Reader Auto Notification Message
#ReaderName: Spinner Reader
#ReaderType: Alien RFID Tag Reader, Model: ALR-9900 (Four Antenna / Gen 2 / 902-928 MHz)
#IPAddress: 10.1.70.13
#CommandPort: 23
#MACAddress: 00:80:66:10:11:6A
#Time: 2003/01/21 12:48:59
#Reason: TEST MESSAGE
The tag 1115 F268 81C3 C012 was read 3 times
The tag 0100 0100 0002 0709 was read 1 times
#End of Notification Message
```

Listening for Tags over the Network

When a reader has been configured for Autonomous Mode, interactive communication with the reader is unnecessary and it can be left to work on its own. It is then up to the application to listen for any notification messages from the reader.

Libraries included in the RFID Reader Developer's Kit provide this functionality in Java, .NET, and Visual Basic environments. In both cases, setting up a listening service is a simple coding task, involving less than ten lines of code.

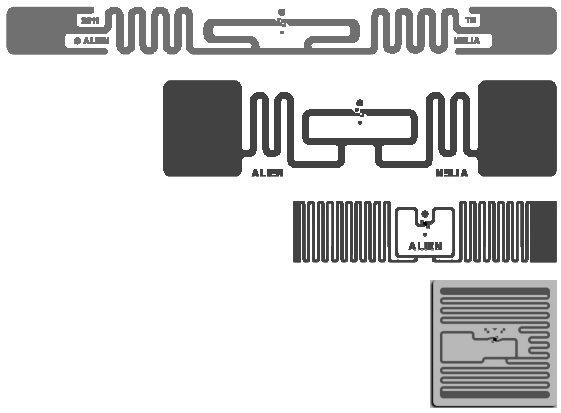
CHAPTER 3
Tag Fundamentals

Introduction

RFID tag reading is not just about getting the tag ID from a tag into the reader. There are different methods available to perform this basic operation, and different parameters and settings that can be altered to tweak the performance of this basic operation.

Alien RFID Tags

Alien Tags



Alien's offerings in the Class 1 tag category use proprietary ICs to meet the cost and size requirements of the EPCglobal specification.

This is a completely "passive" tag by both common definitions of the term in both its means of communication (backscatter) and its power source (beam-powered). Thus, if you hear Aliens refer to a "passive tag," this is the type of tag they are talking about.

Acquisition Modes

The AcquireMode property defines the method used to read tags in the field. There are two distinct methods for reading tags, and the choice of one method over another depends on the application at hand. You specify the Acquire Mode by issuing the "`AcquireMode`" command. It can take one of two values, "Global Scroll" or "Inventory".

Inventory

The inventory command is a full-featured system for discerning the IDs of multiple tags in the field at the same time. This single high-level command transforms itself into a complex series of reader-tag interrogations that eventually resolve themselves into a single list of tag IDs seen by the RFID Reader. This method of interrogation and evaluation of multiple tags is known as an anti-collision search.

CLASS 1/GEN 2 INVENTORY

In a Class 1/Gen 2 inventory, the reader counts the tags in the field in a probabilistic fashion. Since it can't communicate with all tags at the same time, the tags respond in a somewhat random manner, which eventually, though efficiently, scans the entire tag population. The reader tells all of the tags to roll a random number, and the population is subdivided into a number of "bins" (determined by the "Q" parameter). Only those tags that chose a random number in bin #1 get to talk first. The reader attempts to decode all of the responses from the tags in bin #1, and it that group is still too large it divides the tag population into smaller-sized bins. After the reader finishes decoding signals from this first group of tags, it tells the rest of the tags to move down a bin, and it then talks to the next group of tags.

Picking the starting Q parameter that is right for the expected tag population size is important in achieving good inventory performance, but Alien readers can dynamically adjust the Q parameter as the inventory progresses to cause more or fewer tags to respond at any given time, making the initial choice of Q value less important.

CLASS 1/GEN 2 ACQUIRE PARAMETERS

When the reader scans for Class 1/Gen 2 tags, it can automatically prepare tags to be inventoried (called “Selecting”), then perform a series of actions on those tags. The tag population is subdivided into "bins", so that the reader only has to talk to a subset of the tags at any given time. This sequence is defined by the "G2" acquire parameters; by adjusting each value you can configure the reader to perform optimally, depending on your particular use case.

When the reader does an acquire action, it performs a total number of acquire cycles specified by the `AcqG2Cycles` parameter. Each cycle starts by issuing `AcqG2Select` SELECT commands on each antenna, The SELECT command causes some tags to go to the "A" (un-inventoried) state, which is similar to “waking the tag up”. It then performs a number of complete inventories (given by `AcqG2Count`) on each antenna, placing tags into the "B" state as they are read. You control which tags participate through the use of a tag Mask.

The acquire parameters for Class 1/Gen 2 are demonstrated by the following pseudo-code:

```
foreach cycle in AcqG2Cycles
   foreach antenna in AntennaSequence
          SELECT (applies the current mask)
   next antenna

   foreach count in AcqG2Count
      foreach antenna in AntennaSequence
         DoAcquisition
      next antenna
   next count
next cycle
```

Additionally, Class 1/Gen 2 tags can be inventoried in one of four "sessions", specified by the `AcqG2Session` parameter. Each tag maintains a separate "inventoried" state on each session, and once inventoried, a tag does not respond to further reader inquiries on the same session, until it receives another SELECT command. Tags can remain in the inventoried state even after leaving the vicinity of the reader, for a period of time dependent on the session used.

Global Scroll

Global Scroll is the most primitive of tag ID reading operations supported by the Alien RFID Reader system. When a global scroll command is issued, the RFID Reader sends a single command over the air to all and any tags. That command is simply a request for any tag to immediately send back its ID to the RFID Reader.

The simplicity of this command is both its advantage and its downfall: The command is very quick to execute as it involves only one round trip between the reader and the tag. However, because the command is so simple, problems may arise if there is more than one tag in the field. At this point, multiple tags all receive the same command, and all send back their IDs to the reader at virtually the same time. A situation such as this makes it difficult for the reader to discern individual IDs among the general noise. Typically one or two of the loudest or closest tags is decoded, but the majority will not be discerned.

This is analogous to walking into a dark room full of people and shouting out the command “if anyone can hear me, shout your name back now”. If there is one person in the room with you, you are able to hear their name. If there are multiple people in the room, the results will be noise. Maybe you are able to make out one or two names, but typically not more than that.

Furthermore, The Q value is not automatically adjusted during a Global Scroll inventory, and the reader alternates between inventorying Class 1/Gen 2 tags in the “A” state and then in the “B” state, for greater speed. For all of these reasons, Global Scroll mode is generally not recommended for the majority of RFID situations.

Masks and Tag Memory Structure

Many commands aimed at Alien RFID tags require the setting of a mask, which directs the commands only at the tags whose ID matches the mask. This mechanism allows commands to be sent to one specific tag, a selective group of tags, or the whole field of tags.

To understand the use of masks, a basic understanding of tag memory structure is first required.

Class 1/Gen 2 Tag Memory

The basic size of a data entity in Class1, Gen2 is a word (16 bits), rather than a byte (8 bits). Class 1/Gen 2 tags contain up to four memory banks.

G2 Bank	Description
Bank 0 RESERVED	Contains the Kill and Access passwords. Each is two words (4 bytes) in length. Tags need not implement either of these passwords, in which case the values are taken to be 0000 0000, and the corresponding unused memory locations need not exist. Masking into Bank 0 is not permitted.
Bank 1 EPC	Contains one CRC word, one PC word, and a flexible number of EPC words following. See below for detailed structure.
Bank 2 TID	Contains tag identifying information, such as the allocation class identifier (EPCglobal, or other), manufacturer information, and tag model and versioning information.
Bank 3 USER	Unstructured scratch space for user-specific data storage. Tags need not implement the USER bank, and the memory capacity depends on the tag implementation.

96-BIT CLASS 1/GEN 2 STRUCTURE

Current Class 1/Gen 2 tags generally contain 96 or 128 bits of programmable memory in the EPC memory bank, although the specification allows for up to 496 EPC bits. In addition to the EPC code, the EPC memory bank also contains 16 bits of CRC checksum, and 16 Protocol-Control (PC) bits.

	CRC	PC	EPC Code (or User ID Code)						
Word	0	1	0	1	2	3	4	5	…
Bit	0-15	16-31	32-47	48-63	64-79	80-95	96-111	112-127	…

Class 1/Gen 2 Tag Memory Structure

The EPC Code is addressed from left to right, where the leftmost bit (the Most Significant Bit) is bit 32, and extending out through the length of the EPC. There is no restriction on the data that resides in this portion of the tag.

The checksum is automatically calculated by the tag, over the 16 bits of the PC and the entire EPC Code. The checksum is calculated and programmed into the tag automatically by the reader.

The Protocol-Control (PC) word encodes the number of the EPC words, (bits 0-4), followed by two reserved bits (bit 5-6), and a numbering system identifier (bits 7-15). The numbering system identifier

specifies either an EPCglobal™ header (defined by the EPC™ Tag Data Standards) or an Application Family Identifier (defined by ISO/IEC 15961).

For further details on programming tag IDs and tag memory, please see the Tag Programming chapter.

Addressing a Subset of Tags

One of the more useful applications of the mask command is to address a subset of tags in the field. This is achieved using masks.

For example, the following mask command can be issued to address only tag IDs that start with the numbers "8000" :

```
AcqG2Mask = 1, 32, 16, 80 00
```

The first value is the bank (EPC is in bank #1). The second value is the location within the bank where you want to start matching data with the tag (the actual EPC starts at bit #32). The third value is the number of bits in the mask (you might want a length less than 8 bits). Finally, a sequence of hex bytes is given specifying the mask – 80 00.

Subsequent commands that use a mask will now only be recognized by tags that start with this tag ID. This can be useful if, for example, the reader is scanning food items but is only interested in finding a certain brand of breakfast cereal. By setting the mask to identify only the breakfast cereal tag IDs, any acquire command on the food items only returns the items of interest. This methodology works particularly well when combined with the EPC code strategy, where each product type and manufacturer code use well-defined memory codes that can be masked.

Another example is to search for all tags whose last three bits of a 96-bit EPC code are set to 1. The mask settings for this would be:

```
AcqG2Mask = 1, 125, 3, E0
```

In other words, masking in bank #1 again, and since the 96-bit EPC starts at bit #32, then the last three bits will start at bit #125 (32+96-3). The length is 3 bits, and we want to match a data value of $E0_{hex}$. The mask value is specified as $E0_{hex}$ (11100000_{binary}) and not 07_{hex} (111_{binary}) because 07_{hex} is interpreted as 00000111_{binary}, and when the bit pattern is applied as a mask, the bits are applied from left to right (most-significant-bit to least-significant-bit).

The masking operation is done as part of the SELECT portion of the Class 1/Gen 2 over-the-air transaction, so you must have AcqG2Select set to a value larger than 0. If AcqG2Select=0, then no SELECT commands are issued to the tags during an inventory, and the mask will have no effect.

CHAPTER 4
Alien Reader Protocol

The Alien Reader Protocol (ARP) is a text-based communications protocol for configuring and operating an Alien RFID Reader. This chapter describes the programming interface that links the Alien RFID Reader to the outside world.

For an overview of the reader system and instructions on setting up reader operation via a host computer, see the chapters entitled: *Reader Fundamentals* and *Tag Fundamentals*.

Some Alien readers support the EPCglobal Low Level Reader Protocol (LLRP), which is an alternative to ARP. Refer to chapter 6 for more information about LLRP.

Reader Operation Overview

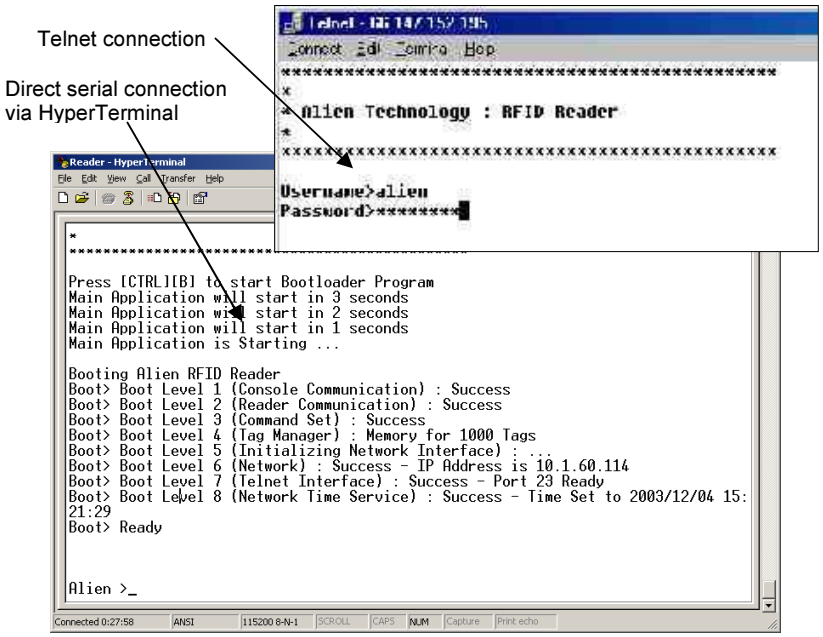
As detailed in the previous chapter, you may interact with the reader and configure its operation using either of two text-based command line methods:

- direct serial (RS-232) – if supported by the reader
- telnet connection (TCP/IP)

For the purposes of these instructions, the serial and telnet operations are considered essentially identical. In both cases, the screens look similar and are considered identical for the purposes of the instructions that follow.

Telnet Exceptions:

- In telnet operation you must issue the command “q” to quit the session.
- Accessing the reader via telnet requires an authorized user name and password (both of which can be changed with commands in the General command set).



Overview of Commands

There are two distinct categories of reader activities: those initiated by the host (Interactive Mode), and those initiated by the reader itself (Autonomous Mode).

INTERACTIVE MODE

Interactive Mode activities are initiated by a program or user who issues commands to the reader from a separate host computer. The host always initiates the transaction with a request, and the reader responds with an immediate reply. The host should wait for a reply before initiating another command.

Interactive commands are used to configure and operate the reader, as well as to interrogate tags and retrieve the stored TagList on demand.

AUTONOMOUS MODE

AutoMode activities instruct the reader to perform certain tasks without outside intervention, according to conditions set by the programmer.

These commands typically tell the reader to read tags based on trigger events (external digital inputs or internal timers) and then send notification messages to a host system, depending on various TagList-based triggers. For example, the reader can be instructed to search the field until it sees a tag, then to read the new tag and e-mail the event to a specified e-mail address.

Command Format

All commands between the host system and the reader are human readable, ASCII text-based messages. To get the value of a particular attribute, use:

```
<attribute>?
```

To set the value of a particular attribute, use:

```
<attribute> = <value>
```

For example, a command to set the name of the reader using the "ReaderName" command takes the form:

```
Alien>ReaderName = My Alien Reader[CR][LF]
```

All commands to the reader are single-line ASCII strings, terminated by a carriage-return / line-feed pair, written as [CR][LF]. The [CR] character is ASCII code 0x0D, while the [LF] character is ASCII code 0x0A. These characters are sometimes written as "\r" and "\n" respectively.

All responses from the reader are either single-line or multi-line ASCII strings, terminated by [CR][LF][0], where [0] is ASCII code 0x00. When a responses is comprised of multiple text lines, each line is separated by a single [CR][LF] sequence.

An example of a command and single-line response is:

```
Alien>ReaderName?[CR][LF]
ReaderName = Alien Reader[CR][LF][0]
```

An example of a command and multi-line response is:

```
Alien>t[CR][LF]
Tag:BEEF 0909 0909 0909, Disc:2008/08/12 ... Count:1, Ant:0, Proto:2[CR][LF]
Tag:DEAD BEEF CAFE 8042, Disc:2008/08/12 ... Count:1, Ant:0, Proto:2[CR][LF]
Tag:DEAD BEEF CAFE 8043, Disc:2008/08/12 ... Count:1, Ant:0, Proto:2[CR][LF][0]
```

Commands are not case sensitive:

"readername?" is equivalent to "ReADerNaME?".

Suppressing Command Prompts

By default, the reader responds to all commands using the interactive console-style interface. Consequently replies are always followed by a command prompt indicating that the reader is ready for more user input. Often, the command prompt is not required, especially when client software is written that programmatically communicates with the reader.

Since the command prompt is sent after the null-terminated response, this prompt text might be interpreted as the beginning of the next reader response. To accommodate this, command prompts can be suppressed by prepending a 0x01 character, written as [1], to the command string. For example:

INTERACTIVE COMMAND FORMAT

```
Alien>ReaderName?[CR][LF]
ReaderName = Alien Reader[CR][LF][0]

[CR][LF]Alien>
```

NON-INTERACTIVE COMMAND FORMAT

```
[1]ReaderName?[CR][LF]
ReaderName = Alien Reader[CR][LF][0]
```

"Get" and "Set" Compatibility

Older versions of the Alien Reader Protocol insisted that you type "get <attribute>" and "set <attribute> = <value>" when getting and setting values, rather than the more convenient "<attribute>?" and "<attribute> = <value>". In order to be backward compatible with systems that still communicate with readers using this older syntax, the reader still accepts that formatting.

Older Syntax	Standard Syntax
Get <attributeName>	<attributeName>?
Set <attributeName> = <attributeValue>	<attributeName> = <attributeValue>

Examples:

```
Alien>get ReaderName
ReaderName = Alien RFID Reader

Alien>ReaderName?
ReaderName = Alien RFID Reader

Alien>set ReaderName = Alice
ReaderName = Alice

Alien>ReaderName = Alice
ReaderName = Alice
```

Command History Buffer, and "!"

The reader remembers the last ten commands that you have entered, and you can scroll through the command history by typing the up- or down-arrow keys. The up-arrow cycles through progressively older commands, while the down-arrow cycles back to the more recent commands.

When a command is retrieved from the history buffer, the cursor is positioned at the end of the line, so you can easily edit the command or just press [ENTER] to send it. This is helpful if you mistype something, or just want to enter another command similar to the previous one.

The "!" command repeats the last command entered. It can also be used to re-set a property with a new value. For instance, after entering "ReaderName=Bob", you could then type "! = Larry" to set the ReaderName to Larry.

XML Messages

There are a few cases where text-based replies and messages are formatted in XML format for easier computer parsing. Complete Document Type Definitions (DTDs) for each XML document are provided in this document as an appendix, as well as on the Developer's Kit CD.

The following messages are sent in XML format:

- Heartbeat Messages
- Notification Messages (if NotifyFormat = XML)
- The "get TagList" commands (if TagListFormat = XML)

Command List

GENERAL COMMANDS

Command	Description	9900	9680	9650
Help ("h")	List all reader commands available.	✓	✓	✓
Info ("i")	List all current reader settings.	✓	✓	✓
! *(exclamation mark)*	Repeat the last command issued.	✓	✓	✓
Save	Save current settings to flash memory	✓	✓	✓
Quit ("q")	Quit session (telnet only)	✓	✓	✓
Function	Used to enable and disable classes of commands (programming, for example)	✓	✓	✓
ReaderName	An arbitrary name to be associated with the reader.	✓	✓	✓
ReaderType	A description of the reader type.	✓	✓	✓
ReaderVersion	Reader software/hardware versions.	✓	✓	✓
DSPVersion	DSP firmware version and configuration.	✓	✓	✓
ReaderNumber	An arbitrary number (1-255) to be associated with the reader.	✓	✓	✓
BaudRate	The serial interface baud rate.	✓	✓	✓
Uptime	Number of seconds that have elapsed since the reader was last booted.	✓	✓	✓
Username	The Username used for the Network based access control.	✓	✓	✓
Password	The Password used for the Network based access control.	✓	✓	✓
MaxAntenna	The maximum addressable antenna port number (total number of antennas is MaxAntenna+1)	✓	✓	✓
AntennaSequence ("ant")	The antenna port sequence the reader will use.	✓	✓	✓
RFAttenuation	The amount of digital attenuation to apply to the emitted RF.	✓	✓	✓
RFLevel	The output power. (RFLevel = <maxPower> – <RFAttenuation>	✓	✓	✓
RFModulation	The Class1/Gen2 modulation scheme.	✓	✓	✓
FactorySettings	Reset the reader to its original factory settings.	✓	✓	✓
Reboot	Reboot the reader.	✓	✓	✓
Service	View status, start/stop, and control auto-starting of auxiliary services.	✓	✓	✓
ETSIMode	Switch the regulatory mode of the ALR-9900-EMA between: EN 302 208 v1.3.1 and EN 300 220	+		
MyData	A general scratch-pad for your personal data.	✓	✓	✓

+ = ALR-9900-EMA only

NETWORK COMMANDS

Command	Description	9900	9680	9650
MACAddress	The Reader's MAC address – an identifier unique to every reader.	✓	✓	✓
DHCP	If DHCP is on, the reader automatically configures its network settings on power-up.	✓	✓	✓
DHCPTimeout	Number of seconds the reader should wait for a reply from a DHCP server.	✓	✓	✓
IPAddress	The network IP address of the reader. If DHCP is enabled this is set automatically.	✓	✓	✓
Gateway	The network Gateway. If DHCP is enabled this is set automatically.	✓	✓	✓
Netmask	The subnet mask. If DHCP is enabled this is set automatically.	✓	✓	✓
DNS	The network Domain Name Server. If DHCP is enabled this is set automatically.	✓	✓	✓
Hostname	The network hostname. May be overridden by the DHCP server.	✓	✓	✓
UpgradeAddress	The address where firmware updates may be retrieved found.	✓	✓	✓
NetworkUpgrade	Enables automatic network firmware upgrades.	✓	✓	✓
UpgradeNow	Triggers a reader firmware update.	✓	✓	✓
NetworkTimeout	The amount of time before the reader closes a stale network socket.	✓	✓	✓
CommandPort	The network port # for the reader's command channel.	✓	✓	✓
CommandPortLocal	The network port # for the reader's local command channel.	✓	✓	✓
AcceptConnections	Controls if reader accepts external TCP connections.	✓	✓	✓
Ping	Tests a network connection to another device.	✓	✓	✓
HeartbeatAddress	The IP address to deliver heartbeat messages to.	✓	✓	✓
HeartbeatPort	The port # where heartbeat messages are directed.	✓	✓	✓
HeartbeatTime	The number of seconds between each heartbeat.	✓	✓	✓
HeartbeatCount	The total number of heartbeats to send after booting.	✓	✓	✓
HeartbeatNow	Triggers an immediate heartbeat message.	✓	✓	✓
WWWPort	The reader's web server port number.	✓	✓	✓
HostLog	Get the log dump of recent host activity.	✓	✓	✓
DebugHost	Controls logging of all host-issued reader commands.	✓	✓	✓

TIME COMMANDS

Command	Description	9900	9680	9650
TimeServer	The address of a network time server.	✓	✓	✓
TimeZone	The time zone offset from UTC for the reader's clock.	✓	✓	✓
Time	The time on the reader's clock.	✓	✓	✓

MACRO COMMANDS

Command	Description	9900	9680	9650
MacroList	Displays the names of all installed macros.	✓	✓	✓
MacroView <macro>	Displays the contents of the named macro.	✓	✓	✓
MacroRun <macro>	Executes the commands in the named macro.	✓	✓	✓
MacroDel <macro>	Permanently removes the named macro.	✓	✓	✓
MacroDelAll	Permanently removes all installed macros.	✓	✓	✓
MacroStartRec <macro>	Creates a new macro, and records subsequent commands into that macro.	✓	✓	✓
MacroStopRec	Halts macro recording.	✓	✓	✓

TAGLIST COMMANDS

Command	Description	9900	9680	9650
get TagList ("t")	Performs and inventory and returns the current list of tags from the reader.	✓	✓	✓
to	Performs an inventory, and also executes AcqG2Ops, regardless of the current AcqG2OpsMode setting.	✓		
PersistTime	The persistence time for tags in the TagList.	✓	✓	✓
TagListFormat	The output format for TagLists.	✓	✓	✓
TagListAntennaCombine	Indicates whether to combine tag reads from different antennas into one TagList entry.	✓	✓	✓
TagListCustomFormat	The custom TagList formatting string.	✓	✓	✓
TagDataFormatGroupSize	The size, in bytes, of data groups in a formatted TagList.	✓	✓	✓
TagListMillis	Displays timestamps in the TagList with millisecond resolution.	✓	✓	✓
Clear TagList	Clear the list of active tags on the reader.	✓	✓	✓
TagStreamMode	Turn Tag Streaming On and Off.	✓	✓	✓
TagStreamAddress	The address (IP:port or "Serial") for streaming Tag events.	✓	✓	✓
TagStreamFormat	The format of the TagStream.	✓	✓	✓
TagStreamCustomFormat	The custom TagStream formatting string.	✓	✓	✓
TagStreamKeepAliveTime	The number of seconds to keep an idle TagStream socket open.	✓	✓	✓
TagStreamCountFilter	Limits amount of streamed tag data.	✓	✓	✓
StreamHeader	Enables a reader-identifying header when opening a TagStream or IOStream socket.	✓	✓	✓

ACQUIRE COMMANDS

Command	Description	9900	9680	9650
AcquireMode	The inventory mode used to reads tags.	✓	✓	✓
AcqG2Cycles	Number of acquisition cycles to perform during each Gen2 tag inventory.	✓	✓	✓
AcqG2Count	Number of reads to perform in each Gen2 inventory cycle.	✓	✓	✓
AcqG2Q	Starting "Q" value to use in each Gen2 inventory.	✓	✓	✓
AcqG2QMax	Maximum value of Q parameter during Gen 2 reads.	✓	✓	✓
AcqG2Select	How many times a B->A Select is issued at the start of a Gen2 inventory.	✓	✓	✓
AcqG2Session	The G2 Session to use when inventorying tags.	✓	✓	✓
TagType	Specifies which tag types to look for.	✓	✓	✓
G2Wake	Issue a Class1/Gen2 Select command to force tags into the un-inventoried state.	✓		
AcqG2Mask	The Class1/Gen2 mask. Any memory bank.	✓	✓	✓
AcqG2MaskAction	Whether the AcqG2Mask includes or excludes matching tags.	✓	✓	✓
AcqG2MaskAntenna	The bitmap of antennas where the G2 mask applies.	✓		
AcqG2SL	Controls whether the G2 SL flag is used during inventories.	✓		
AcqG2AccessPwd	The Access password used to operate on password-protected Gen2 tags.	✓	✓	✓
AcqG2TagData	Specify additional Class1/Gen2 tag data blocks to retrieve for the TagList.	✓		
AcqG2Target	Set the Class1/Gen2 inventory target (A, B, A<->B)	✓	✓	✓
AcqG2AntennaCombine	Controls how the Gen2 SELECT command is issued during the inventory cycle	✓		
AcqG2Ops	Set additional C1G2 tag operations, to be performed during a G2Ops tag inventory.	✓		
AcqG2OpsMode	Enables AcqG2Ops operations in all inventories.	✓		
AcqTime	Limits duration of the inventory	✓		
SpeedFilter	Filters out tags based on their speed and direction.	✓		
RSSIFilter	Filters out tags based on their RF signal strength.	✓	✓	✓
TagAuth	Alien Dynamic tag Authentication filter	+		

+ = ALR-9900+ only

I/O COMMANDS

Command	Description	9900	9680	9650
ExternalInput	Gets the External Input values.	✓	✓	✓
ExternalOutput	Gets and sets the External Output values.	✓	✓	✓
InvertExternalInput	Turn On or Off the inversion of the External Inputs. When inverted, driving an input voltage high indicates low.	✓	✓	✓
InvertExternalOutput	Turn On or Off the inversion of the External Outputs. When inverted, setting an output high drives its voltage low.	✓	✓	✓
InitExternalOutput	The External Output state the apply during and after the reader boots.	✓	✓	✓
IOList ("ios")	Get a formatted list of I/O events.	✓	✓	✓
Clear IOList	Clears all I/O events from the IOList.	✓	✓	✓
IOType	The type of I/O events to track.	✓	✓	✓
IOListFormat	The format of the IOList.	✓	✓	✓
IOListCustomFormat	The custom IOList formatting string.	✓	✓	✓
IOPersistTime	The persistence time for IO events in the IOList.	✓	✓	✓
IOStreamMode	Turns I/O Streaming On and Off.	✓	✓	✓
IOStreamAddress	The address (IP:port or "Serial") for streaming I/O events.	✓	✓	✓
IOStreamFormat	The format of the I/O Stream.	✓	✓	✓
IOStreamCustomFormat	The custom I/O Stream formatting string.	✓	✓	✓
IOStreamKeepAliveTime	Number of seconds to keep an idle IOStream socket open.	✓	✓	✓
BlinkLED	Blinks the reader's status LEDs in a user-defined way.	✓	✓	✓

AUTOMODE COMMANDS

Command	Description	9900	9680	9650
AutoMode	Switch auto mode on and off.	✓	✓	✓
AutoWaitOutput	The value of the outputs while in the wait state.	✓	✓	✓
AutoStartTrigger	The input trigger that starts the AutoAction.	✓	✓	✓
AutoStartPause	Milliseconds after receiving a start trigger before starting the AutoAction	✓	✓	✓
AutoWorkOutput	The value of the outputs while in the work state.	✓	✓	✓
AutoAction	The action to perform in AutoMode's work state.	✓	✓	✓
AutoStopTrigger	The input trigger that stops the AutoAction.	✓	✓	✓
AutoStopTimer	Milliseconds to perform the AutoAction.	✓	✓	✓
AutoStopPause	Milliseconds after receiving a stop trigger before stopping the AutoAction.	✓	✓	✓
AutoTrueOutput	The value of the outputs when AutoMode evaluates to True.	✓	✓	✓
AutoTruePause	Milliseconds to pause after AutoMode evaluates to True.	✓	✓	✓
AutoFalseOutput	The value of the outputs when AutoMode evaluates to False.	✓	✓	✓
AutoFalsePause	Milliseconds to pause after AutoMode evaluates to False.	✓	✓	✓
AutoErrorOutput	The error-dependent value of the outputs when AutoMode evaluates to False.	✓		
AutoProgError	Set the error code to report when AutoMode programming action fails.	✓		
AutoModeTriggerNow	Force an AutoStartTrigger event to occur.	✓	✓	✓
AutoModeReset	Reset all AutoMode values to their default states.	✓	✓	✓

NOTIFY COMMANDS

Command	Description	9900	9680	9650
NotifyMode	Switches notify mode On and Off.	✓	✓	✓
NotifyFormat	The format for TagList notification messages.	✓	✓	✓
NotifyHeader	Turns On and Off the additional header (and footer) lines in notification messages.	✓	✓	✓
NotifyAddress	The address to push TagList notifications to.	✓	✓	✓
NotifyTime	Number of seconds between timer-triggered notification messages.	✓	✓	✓
NotifyTrigger	The AutoMode-generated trigger for pushing TagLists.	✓	✓	✓
NotifyKeepAliveTime	Number of seconds to keep an idle notification socket open.	✓	✓	✓
MailServer	The SMTP mail server for e-mail notifications.	✓	✓	✓
MailFrom	The email address identifier of the RFID Reader.	✓	✓	✓
NotifyRetryCount	The number of times a failed socket notification attempt is repeated.	✓	✓	✓
NotifyRetryPause	Number of seconds between failed notification retries.	✓	✓	✓
NotifyQueueLimit	Number of failed notifications to queue for later delivery.	✓	✓	✓
NotifyInclude	Whether notifications should include Tag data, I/O event data, or both.	✓	✓	✓
NotifyNow	Forces a message via the notification system.	✓	✓	✓

Commands related to programming tag memory are covered in their entirety in a later chapter.

The following sections describe each command category - detailing each command, its use and the response formats.

*NOTE: RFID Reader commands are **not** case sensitive, that is, you can use upper or lower case, or any combination thereof, and the reader will understand the command. Capitalization of commands is used in this document and in actual command responses solely for the purpose of readability.*

General Commands

General commands cover basic reader, antenna functions and help & information.

Help (h)

9900 | 9680 | 9650

This command lists all reader commands available, categorized by function. You may also enter a command name as an argument to the Help command, and receive help text for just that command.

- The command "h" is a shortcut for the "Help" command.

Help Examples	
Command Response	>help ********** GENERAL COMMANDS ********** Help: Display descriptive help text. Info: Display current settings. !: Repeat last command. Save: Save current settings to permanent storage. ...
Command Response	// Help for single command >h NotifyNow NotifyNow: Manually trigger NotifyMode.

Info (i)

9900 | 9680 | 9650

This provides a list of current reader settings. Reader settings are categorized by function.

- The command "i" is a shortcut for the "Info" command.

You may also enter a command group name as an argument to the Info command, in order to just see the current settings belonging to that group. Group names appear surrounded by rows of asterisks in the Help and Info dumps.

Info Examples	
Command Response	>Info ********** GENERAL COMMANDS ********** Function = Reader ReaderName = Alien RFID Reader ReaderType = Alien RFID Tag Reader, Model: ALR-9800... ReaderVersion = 07.01.10.00 DSPVersion = DSP:02.01.05 reader:05 country:01 radio:32 board:01 ReaderNumber = 255 ...
Command Response	// Info for a single command group >i network ********** NETWORK COMMANDS ********** MACAddress = 00:80:66:10:11:6A DHCP = ON IPAddress = 10.10.82.72 Hostname = fubar UpgradeAddress = 0.0.0.0 NetworkUpgrade = OFF ...

!

9900 | 9680 | 9650

This command (exclamation mark) asks the reader to repeat the last command issued.

! Example	
Command Response	>Program Tag = DE AD BE EF CA FE B0 B0 Error 134: No Tag Found. >! Program Tag = Success!

You can also repeat a previous "set" command with a new value using the "! =" shortcut. If the previous command wasn't a "set" command, the reader responds with a Malformed Message error.

"! =" Example	
Command Response Command Response	>ExternalOutput = 0 ExternalOutput = 0 >! = 1 ExternalOutput = 1
Command Response Command Response	>ReaderName? ReaderName = myAlienReader >! = yourAlienReader Error: Malformed Message

Save

9900 | 9680 | 9650

The Save command writes the current settings to flash memory. This ensures that if the reader loses power it restarts in the same state.

Save Example	
Command	>Save
Response	All settings have been saved to flash!

Quit (q)

9900 | 9680 | 9650

(For Telnet operation only) The Quit command allows you to exit the current Telnet session.

- The command "q" is a shortcut for the "Quit" command.

Function

9900 | 9680 | 9650

The Function command is supported for backward-compatibility with older reader models. In older readers, the programming functions (program, lock, erase, kill, etc.) had to first be enabled by changing the Function property from "Reader" to "Programmer". Current readers always have programming functionality enabled, so this command is no longer required.

- Allowed Values: "Reader" | "Programmer"
- Default Value: "Reader"

Function Examples	
Command	> Function?
Response	Function = Reader
Command	>Function = Programmer
Response	Function = Programmer
Command	>Function = Reader
Response	Function = Reader

ReaderName

9900 | 9680 | 9650

The reader can be assigned an arbitrary text name to aid identification in multiple-reader environments. This name can be retrieved and changed at any time throughout reader operation.

- Allowed Values: String[1-254]
- Default Value: "Alien RFID Reader"

ReaderName Examples	
Command	>ReaderName?
Response	ReaderName = My First Alien Reader
Command	>ReaderName = My Second Alien Reader
Response	ReaderName = My Second Alien Reader

ReaderType

9900 | 9680 | 9650

The reader type can be retrieved using this command. The resulting text is a single-line reply describing the model number of the reader and related information.

ReaderType Example	
Command	>ReaderType?
Response	ReaderType = Alien RFID Tag Reader, Model: ALR-9900 (Four Antenna / Gen 2 / 902-928 MHz)
Command	>ReaderType?
Response	ReaderType = Alien RFID Tag Reader, Model: ALR-9650 (One Antenna / Gen 2 / 902-928 MHz)

ReaderVersion

9900 | 9680 | 9650

The reader version can be retrieved using this command. The resulting text is a multi-line reply. Each line of the reply describes the version number of a major reader component, as well as some additional locality information.

ReaderVersion Example	
Command	>ReaderVersion?
Response	ReaderVersion = 06.06.18.00

DSPVersion

9900 | 9680 | 9650

The DSP version can be retrieved using this command. It also displays configuration and locality information that may be useful to Alien Support in troubleshooting matters.

DSPVersion Examples	
	// ALR-9900
Command	>DSPVersion?
Response	DSPVersion = DSP:04.06.01 reader:08 country:01 radio:35 board:04 FPGA:09.03.17.00
	// ALR-9680 & 9650
Command	>DSPVersion?
Response	DSPVersion = DSP:02.003.032-01.003.000 reader:09 country:01 radio:12 board:20

ReaderNumber

9900 | 9680 | 9650

The reader can be assigned an arbitrary number to aid identification in multiple-reader environments. This number can be retrieved and changed at any time throughout reader operation.

- Allowed Values: Integer(1..255)
- Default Value: 255

ReaderNumber Examples	
Command	>ReaderNumber?
Response	ReaderNumber = 255
Command	>ReaderNumber = 15
Response	ReaderNumber = 15

BaudRate

9900 | 9680 | 9650

The serial interface to the reader normally operates at 115,200 baud. This is the highest data rate that is standard with most serial port implementations. The reader's serial baud rate can be changed to one of four slower rates, in order to accommodate legacy equipment.

- Allowed Values: 9600 | 19200 | 38400 | 57600 | 115200
- Default Value: 115200
- The values 0..4 are also accepted (they each map to one of the actual baud rate values above), in order to remain backward compatible with older reader models.
- Changes do not take effect immediately - the reader must be rebooted in order for the BaudRate change to take effect.

BaudRate Examples	
Command Response	>BaudRate = 19200 BaudRate = 19200

Uptime

9900 | 9680 | 9650

The Uptime command returns the elapsed time, in seconds, since the last time the reader was rebooted.

Uptime Example	
Command Response	>Uptime? Uptime (secs) = 702048

Username

9900 | 9680 | 9650

The reader can be operated over the network. When operated in this mode it uses a simple username/password authentication scheme to stop unwelcome visitors accessing it. This command allows the username to be defined and obtained.

- Allowed Values: String[1-80]
- Default Value: "alien"
- A username/password pair is not required when operating the reader via serial connection.
- NOTE: The username is case sensitive, and should only contain visible ASCII characters.

Username Examples	
Command Response	>Username? Username = alien
Command Response	>Username = hal Username = hal

Password

9900 | 9680 | 9650

The reader can be operated over the network. When operated in this mode it uses a simple username/password authentication scheme to stop unwelcome visitors accessing it. This command allows the password to be defined and obtained.

- Allowed Values: String[1-80]
- Default Value: "password"
- A username/password pair is not required when operating the reader via serial connection.
- NOTE: The password is case sensitive, and should only contain visible ASCII characters.

Password Examples	
Command Response	>Password? Password = password
Command Response	>Password = 1234fab Password = 1234fab

MaxAntenna

9900 | 9680 | 9650

To determine the maximum addressable antenna port of the reader, use the `MaxAntenna` command. Antenna ports are numbered starting at zero, so the actual number of ports is one more than the MaxAntenna value. A reader with four antenna ports returns a MaxAntenna of 3. Similarly, a reader with only two antenna ports returns a MaxAntenna of 1.

MaxAntenna Example	
Command Response	>MaxAntenna? MaxAntenna = 3

AntennaSequence (ant)

9900 | 9680 | 9650

The reader can support the use of multiple antennas. This command allows you to select which antenna port(s) to use and in what sequence. You may also choose to specify a single antenna in the AntennaSequence.

The reader cycles through all antennas, in the order specified in the AntennaSequence for each atomic air protocol command. This includes Sleeps, Wakes, and Selects, in addition to normal inventory commands.

- Allowed Values: 1-8 Integers (each, 0..3)
- Default Value: "0 1"
- The command "ant" is a shortcut for the "AntennaSequence" command.
- Up to eight antenna port values may be specified in the sequence. Antenna ports may be listed multiple times, to cause the reader to spend more time on those antennas.

Mono-Static AntennaSequence Examples	
Command Response	>AntennaSequence? AntennaSequence = 0
Command Response	// To always use antenna 1: >AntennaSequence = 1 AntennaSequence = 1
Command Response	// To cycle between antenna 0 and antenna 1: >AntennaSequence = 0 1 AntennaSequence = 0 1
Command Response	// To weight antenna 0 more than antenna 1: >AntennaSequence = 0 0 0 1 AntennaSequence = 0 0 0 1

RFAttenuation

9900 | 9680 | 9650

Alien RFID readers output up to 1 watt of RF power at each antenna. While this power is sufficient to provide good penetration and range, these attributes are not always desirable. If multiple readers are in the same vicinity, their signals may interfere with each other. Also, in situations where tagged products are close together, but only one product should be read at a time (a conveyor belt, for example), then penetrating power and long range are your enemies.

Attenuating RF reduces its power, and there are two ways to do this. The first method involves placing attenuators inline in the antenna cable. This method is quick but is not flexible and, more importantly, reduces both the emitted RF power as well as the return signal from the tag, which is already very weak. This would impair the reader's ability to detect tags.

The second method uses software-controlled digital attenuation, built into Alien readers. Using the software-controlled digital attenuation reduces the emitted power but not the return signal. The RFAttenuation value ranges from 0 (no attenuation, maximum power) up to maxAttenuation (maximum attenuation, minimum power), in increments of 10 - each "decade" representing an additional 1 dB of RF attenuation.

The command could be used in either of the following variations:

1. `RFAttenuation = <attenuation>`

   This command sets the same attenuation for all antennas.

2. `RFAttenuation = <antenna> <attenuation>`

   This command sets attenuation for each antenna individually.

3. `RFAttenuation = <antenna> <readattenuation> <writeattenuation>`

`<antenna>`	is the antenna number.
`<readattenuation>`	is the RF attenuation to use when reading tags as well as when reading/singulating tags during programming.
`<writeattenuation>`	is the RF attenuation to use when writing to the tag.

   This command sets RF attenuations for read-power-during-programming and write-power-during-programming to better control the programming environment. RF attenuations are specified for each antenna independently.

For example, to keep the program zone small you could read/singulate tags at low power (high RF attenuation) but, once the tag is singulated, use full power (no RF attenuation) to write to the tag. This ensures there is always enough power to complete the write operation.

NOTE: This variation of the `RFAttenuation` command is not available on the ALR-9680 and ALR-9650.

The value of the `<attenuation>` parameter is as follows:

- Allowed Values: Integer(0..150) (may depend on locality)
- Default Value: 0 (10 for ALR-9680 & 9650)
- maxAttenuation depends on how the reader's radio has been calibrated, but is generally 150 (corresponding to 15 dB).
- Increasing RFAttenuation by 10 reduces RF power by 1 dBm.
- The RFAttenuation value applies to all antennas, unless the command is used with an optional antenna argument. See examples below.

RFAttenuation Examples	
Command Response	>RFAttenuation? RFAttenuation = 0
Command Response	// To reduce power by 3 dB: >RFAttenuation = 30 RFAttenuation = 30

Command
Response
 	
Command
Response
 	
Command
Response
 	
Command
Response
	// Set attenuation for only antenna 0
	>RFAttenuation = 0 30
	RFAttenuation = 30
	
	// Compare attenuation of ant 0 with ant 1
	>RFAttenuation 0?
	RFAttenuation = 30
	
	>RFAttenuation 1?
	RFAttenuation = 0

Command
Response
 	
Command
Response
	// get RFAttenuation (without antenna) always gives ant 0 value
	>RFAttenuation?
	RFAttenuation = 30
	
	// read/singulate tags at low power (attenuation of 100 or 10 dB)
	// and program at high power (0 dB attenuation)
	>RFAttenuation = 0 100 0
	RFAttenuation = 0 100 0

RFLevel

9900 | 9680 | 9650

Rather than setting the RFAttenuation (amount of power reduced from full-power), you may instead specify and retrieve the actual power output of the reader. This is for convenience only, as the two commands ultimately control the same radio hardware.

RFLevel and RFAttenuation are always related by the formula: MaxPower = RFLevel + RFAttenuation

The command could be used in either of the following variations:

1. `RFlevel = <power>`This command sets the same power level for all antennas.

2. `RFlevel = <antenna> <power>`
   This command sets power level for each antenna individually.

3. `RFlevel = <antenna> <readpower> <writepower>`

   `<antenna>` is the antenna number.
   `<readpower>` is the RF power level to use when reading tags as well as when reading/singulating tags during programming.
   `<writepower>` is the RF power level to use when writing to the tag.

   This command sets power levels for read-power-during-programming and write-power-during-programming to better control the programming environment. Power settings are specified for each antenna independently.
   For example, to keep the program zone small you could read/singulate tags at low power but, once the tag is singulated, use full power to write to the tag. This ensures there is always enough power to complete the write operation.

   NOTE: This variation of the `RFLevel` command is not available on the ALR-9580 & 9650.

The value of the `<power>` parameter is as follows:

- Allowed Values: Integer(<minPower>..<maxPower>) (depends on locality)
- Default Value: <maxPower>
- The minimum value for RFLevel is MaxPower-MaxAttenuation.
- Increasing RFLevel by 10 increases RF power by 1 dB, and you see a corresponding decrease by 10 in the RFAttenuation.
- The RFLevel value applies to all antennas, unless the command is used with an optional antenna argument. See examples below.

RFLevel Examples

Command Response	>RFLevel? RFLevel = 316
 Command Response	// To reduce power by 3 dB: >RFLevel = 286 RFLevel = 286

Command
Response
 	
Command
Response
 	
Command
Response
 	
Command
Response
 	
Command
Response
	// Set RFLevel of only antenna 0
	>RFLevel = 0 250
	RFLevel = 250
	
	// Compare RFLevel of ant 0 with ant 1
	>RFLevel 0?
	RFLevel = 250
	
	>RFLevel 1?
	RFLevel = 316
	
	// Get RFLevel (without arguments) always gives ant 0 value
	>RFLevel?
	RFLevel = 250
	
	// read/singulate tags at low power (200) and program at high power (300)
	>RFLlevel = 0 200 300
	RFLlevel = 0 200 300

RFModulation

9900 | 9680 | 9650

The RFModulation command allows you to select from several modulation modes available with the Gen2 protocol. Each mode selects a pre-defined set of Gen2 air protocol parameters, and an internal inventory algorithm to best suit a particular circumstance.

- Allowed Values: "STD" | "HS" | "DRM" | "25FM0" | "06FM0" | "25M4" | "12M4" | "06M4" (depends on locality)
- Default Value: "DRM" (depends on locality)

Three of the modes are convenient names for the other defined modes:

- DRM – Dense Reader Mode (same as 25M4)
- STD – Standard (same as 25FM0)
- HS – High Speed (same as 06FM0)
- 25FM0 – 25 µsec Tari, FM0
- 06FM0 – 06 µsec Tari, FM0
- 25M4 – 25 µsec Tari, Miller4
- 12M4 – 12 µsec Tari, Miller4
- 06M4 – 06 µsec Tari, Miller4

Some reader models, or country-specific versions of those models, may not support all modulation modes, due to regulatory restrictions. Also, the ALR-9650/9680 readers only support Dense Reader Mode (DRM).

Value	Description
STD	Standard operating mode. This mode is good for general purpose use. It combines a good mix of speed and noise/interference immunity. This mode meets the "EPCglobal Dense Reader Mode" spectral mask requirements.
HS	High Speed operating mode. Uses high reader-to-tag data rates for better performance with small tag populations, in environments with few readers.
DRM	Dense Reader Mode (default). Enhanced filtering for better performance in "noisy" environments. Recommended when many readers are operating in the same area. This mode meets the "EPCglobal Dense Reader Mode" spectral mask requirements.
[Tari][Mod]	All of the remaining modes follow a standard naming convention: the Tari (in µsec), followed by the modulating scheme, such as FM0, M4 (Miller 4), etc.

RFModulation Examples	
Command	`>RFModulation?`
Response	`RFModulation = DRM`
Command	`>RFModulation = HS`
Response	`RFModulation = HS`

FactorySettings

9900 | 9680 | 9650

The FactorySettings command resets all reader settings to their factory default values. The ALR-x780 readers then reboot themselves. The reboot is required in order for the new settings (especially network values) to take effect.

FactorySettings Examples	
Command	>FactorySettings
Response	All settings reset to factory defaults.

Reboot

9900 | 9680 | 9650

The Reboot command causes the reader to immediately close all network connections and reboot.

- It is recommended that you reboot the reader any time network configuration parameters are changed.
- Readers may also be rebooted via their web interface. Browse to the reader's IP address in a web browser, click the Manage Reader link and follow the provided instructions.

Reboot Examples	
Command	>Reboot
Response	Rebooting System...

Service

9900 | 9680 | 9650

You use the Service command to control auxiliary reader services, such as the serial interface and SNMP. The service command is followed by the name of the service you wish to control, followed by a command. The "all" keyword may be used in place of a service name, to control all installed services.

```
Service <servicename> <command>
```

The current list of services is:

Service	Description
serial	This controls the reader's serial interface. When the serial interface is turned off, the serial port connects into the reader's underlying Linux operating system prompt, which is intended for Alien use only.
snmp	Readers implement the Simple Network Management Protocol v1, MIB2. If you do not have a use for this service, turning it off frees up some system resources.
heartbeat	The reader's heartbeat service can now be completely shut off or disabled – independent of "softer" ways of doing so, by setting HeartbeatCount=0, for instance. Stopping and disabling the heartbeat service prevents it from ever starting up again – even after a firmware upgrade.
ifmon	The reader's service that monitors the Ethernet interface (every 5 minutes) as follows: if DHCP is ON and either Ethernet is DOWN or the DHCP lease has expired, it renews the DHCP lease (or sets a static IP if the DHCP renew fails); if DHCP is OFF and Ethernet is DOWN, then it sets a static IP. When setting a static IP, the service uses 192.168.1.100/255.255.255.0 or the last static IP/netmask values that were used (if any).

The available service commands are:

Command	Description
status	Reports the current status of the selected service, or lists the status of all installed services if "all" is given as the service name. Each line lists the service name, whether it is currently running (R = running, s = stopped), whether autostart is enabled (A = enabled, d = disabled), and a number indicating the startup order of the services.
start **stop**	Starts or stops the service. This controls the immediate state of the service only. If autostart is enabled, the service starts up again the next time that the reader reboots. The reader responds with the new status line for the service(s).
enable **disable**	Enables or disables the autostart feature for the service. When autostart is enabled, the service starts up automatically when the reader boots. Changing the autostart setting does not immediately start or stop the service – use the start and stop commands to accomplish this. The reader responds with the new status line for the service(s).

The "start" and "stop" service commands act right away, but don't change which services are automatically started up if the reader reboots. The "enable" and "disable" service commands affect how the reader starts up on subsequent reboots, but they don't change the which services are running right now.

The start/stop and enable/disable settings of individual services do not need to be "saved", as other reader properties do.

Service Examples	
Command Response	>service serial status R A 05 serial // currently **R**unning, **A**utostart enabled, startup priority **5**
Command Response	>service all status R A 05 serial R A 80 snmp
Command Response	>service serial stop s A 05 serial // the serial service is **s**topped, but it starts up again on the next boot
Command Response	>service serial disable s d 05 serial // service is **d**isabled, it will not be started on the next boot

MyData

9900 | 9680 | 9650

The MyData command gives you a 254-character scratchpad to store whatever data you wish. Some examples might be the reader location with GPS coordinates or the reader provisioning status or history.

- You must issue the Save command to make your MyData changes persistent.
- MyData content is retained across firmware upgrades, like other reader properties.

MyData Examples	
Command Response	>MyData? MyData = (Not Set)
Command Response	>MyData = It's warm and sunny here: Lat: 37.14276, Long: -121.65615 MyData = It's warm and sunny here: Lat: 37.14276, Long: -121.65615

ETSIMode

9900+ | 9900 | 9680 | 9650

The default ETSI mode that the reader operates under is EN 302 208 v1.3.1 for the ALR-9900+ reader. The ALR-9900+ can be directed to meet the requirements of the EN 300 220 standard using the ETSIMode command.

- Allowed Values: "302.208v1.3.1" or "300.220"
- Default Value: "302.208v1.3.1"

The reader must be rebooted in order for changes to the ETSIMode to take full effect. After the ETSIMode is changed, the ReaderType reflects the new regulatory standard.

ETSIMode Examples	
Command Response	>ETSIMode? ETSIMode = 302.208
Command Response	>ReaderType? ReaderType = Alien RFID Tag Reader, Model ALR-9900+ (Four Antenna / Gen 2 / EN 302 208)
Command Response	>ETSIMode = 300.220 ETSIMode = 300.220
Command Response	>ReaderType? ReaderType = Alien RFID Tag Reader, Model ALR-9900+ (Four Antenna / Gen 2 / EN 300 220)

Network Configuration Commands

These commands allow you to configure and retrieve settings related to reader communications with the network.

MACAddress

9900 | 9680 | 9650

The MAC (Media Access Control) Address is a unique, hardcoded value that identifies each device with a network interface. The reader's MAC Address can be retrieved with the `MACAddress` command.

The value returned is a sequence of six hex bytes, separated by colons.

MACAddress Example	
Command	>MACAddress?
Response	MACAddress = 00:90:c2:c3:14:38

DHCP

9900 | 9680 | 9650

The reader supports automatic network configuration using the widely available DHCP protocol. If DHCP is available at the reader installation site, this protocol can be switched on. If DHCP is not available or not desired the use of this protocol should be switched off.

- Allowed Values: "ON" | "OFF"
- Default Value: "ON"
- After making changes with this command, you must issue the `Save` command and reboot the reader to implement the changes.
- If DHCP is on and no server is available when the reader boots up, it will use a default network address (192.169.1.100), and then periodically try to communicate with the DHCP server.

DHCP Examples	
Command	>DHCP?
Response	DHCP = ON
Command	>DHCP = OFF
Response	DHCP = OFF

DHCPTimeout

9900 | 9680 | 9650

When the reader is set up to get it's network configuration automatically from a DHCP server, it gives up waiting for a reply from a DHCP server after a certain period of time, and use a default (static) network configuration instead (IP: 192.168.1.100, Netmask: 255.255.255.0, etc.) The amount of time to wait for the DHCP server's reply is given by the DHCPTimeout property.

The default value, 90 seconds, is generally fine for most situations, but it has been found that on networks with high latencies or when the reader is plugged directly into a LAN drop (not plugged into a router or network switch), a higher timeout works better. If you find the reader is often not getting its network

configuration settings from the DHCP server when it boots, you may want to increase the DHCPTimeout to 120 seconds or higher.

- Allowed Values: Integer(0..1000)
- Default Value: 90
- After making changes with this command, you must issue the `Save` command and reboot the reader to implement the changes.

DHCPTimeout Examples	
Command Response	>DHCPTimeout? DHCPTimeout = 90
Command Response	>DHCPTimeout = 120 DHCPTimeout = 120

IPAddress

9900 | 9680 | 9650

If DHCP is not used for automatic configuration, the reader must be manually configured for use on a network. The IPAddress command allows you to assign and retrieve the host's IP address.

- DHCP should be off in order to change the IPAddress.
- After making changes with this command, you should issue the `Save` command and reboot the reader to implement the changes.

IPAddress Examples	
Command Response	>IPAddress? IPAddress = 12.34.56.78
Command Response	>IPAddress =34.55.33.12 IPAddress = 34.55.33.12

Gateway

9900 | 9680 | 9650

If DHCP is not used for automatic configuration, the reader must be manually configured for use on a network. The gateway command allows the network gateway to be assigned and retrieved.

- Gateway must be specified as a numerical IP address.
- DHCP should be off in order to change the Gateway.
- After making changes with this command, you should issue the `Save` command and reboot the reader to implement the changes.
- If the Gateway address doesn't point to a valid address on your network, then the reader may not be able to make any outbound connections (e.g. TagStream, Heartbeats, Notification messages).

Gateway Examples	
Command	>Gateway?
Response	Gateway = 34.56.78.90
Command	>Gateway=12.56.23.01
Response	Gateway = 12.56.23.01

Netmask

9900 | 9680 | 9650

If DHCP is not used for automatic configuration, the reader must be manually configured for use on a network. The subnet mask command allows the subnet mask to be assigned and retrieved.

- A subnet mask must be specified as a numerical IP address.
- DHCP should be off in order to change the Netmask.
- After making changes with this command, you should issue the `Save` command and reboot the reader to implement the changes.

Netmask Examples	
Command	>Netmask?
Response	Netmask = 255.255.255.128
Command	>Netmask=255.255.255.0
Response	Netmask = 255.255.255.0

DNS

9900 | 9680 | 9650

If DHCP is not used for automatic configuration, the reader must be manually configured for use on a network. The DNS command allows the DNS server location to be assigned and retrieved.

- A DNS server must be specified as a numerical IP address.
- DHCP should be off in order to change the DNS.
- After making changes with this command, you should issue the `Save` command and reboot the reader to implement the changes.

DNS Examples	
Command	>DNS?
Response	DNS = 12.34.56.78
Command	>DNS=45.224.124.34
Response	DNS = 45.224.124.34

Hostname

9900 | 9680 | 9650

You can configure the reader's identity on the network with the Hostname command. When a reader's hostname is known to a local Domain Name Server, that reader may be accessed over the network by it's hostname, without having to know its IP address.

It is common for a DHCP server to assign hostnames to network devices, and in these cases the reader uses whatever hostname is supplied by DHCP, even if you have specified a different hostname with the Hostname command.

- Allowed Values: String[1-16]
- Default Value: "alien-xxyyzz", where xx, yy, and zz are the last three bytes of the reader's unique MAC address.
- Changes to the hostname take effect immediately.

Hostname Examples	
Command	>Hostname?
Response	Hostname = alien-10116A
Command	>Hostname = reader1
Response	Hostname = reader1

NetworkUpgrade

9900 | 9680 | 9650

Alien readers can be configured to automatically upgrade themselves at bootup. After configuring the UpgradeAddress, you enable this feature by setting the NetworkUpgrade flag to On, then simply reboot the reader. You can alternatively issue the UpgradeNow command to trigger an upgrade check immediately.

When the reader boots up, it checks the NetworkUpgrade flag to determine if it should look for an upgrade. If so, it attempts to pull down an upgrade file from an FTP, HTTP, or HTTPS server. It then installs the upgrade, and continues the bootup process.

Details on the various upgrade mechanisms can be found in Appendix C, at the end of this document.

- Allowed Values: "ON" | "OFF"
- Default Value: "OFF"
- Set NetworkUpgrade to "On", issue the "Save" command, and reboot the reader to initiate a network upgrade.

NetworkUpgrade Examples	
Command	>NetworkUpgrade?
Response	NetworkUpgrade = Off
Command	>NetworkUpgrade = On
Response	NetworkUpgrade = On

UpgradeAddress

9900 | 9680 | 9650

You can use the UpgradeAddress command to specify the network host where the reader looks for upgrades. The value specified is a complete URL to the HTTP, HTTPS, or FTP server and directory path where the upgrade files are located. The reader attempts to download a "control" file at the path specified, which contains the name of the upgrade file to download. Complete details of this process are found in Appendix C.

- The default value for UpgradeAddress is 0.0.0.0.
- Readers accepts the "UpgradeIPAddress" command, for compatibility with older readers.

- Changes to the UpgradeAddress commands take effect (after issuing the Save command) upon the next reboot.

UpgradeAddress Examples	
Command Response	>UpgradeAddress? UpgradeAddress = 0.0.0.0
Command Response	>UpgradeAddress = http://192.168.1.200/alienUpgrades/ UpgradeAddress = http://192.168.1.200/alienUpgrades/

UpgradeNow

9900 | 9680 | 9650

The UpgradeNow command allows you to trigger a firmware upgrade without having to reboot your reader. See the sections on NetworkUpgrade, UpgradeAddress (above), as well as the "Pulling Upgrades" section of appendix C for more information about the network upgrade feature.

The easiest way to use UpgradeNow is to preset the UpgradeAddress with the URL where the firmware and control files are at, and then simply issue the UpgradeNow command. You can alternatively include a URL as an argument to the UpgradeNow command and the reader looks there instead.

```
UpgradeNow
```

or

```
UpgradeNow http://192.168.100.150/firmware/
```

You can also include a "list" argument to the command and the reader looks at what is at the UpgradeAddress (or URL you provide) and lists the files found there.

```
UpgradeNow list
```

or

```
UpgradeNow list http://192.168.100.150/firmware/
```

Finally, there is a "force" argument which causes the reader to download and install whatever firmware it finds, regardless of the current firmware version.

```
UpgradeNow force
```

or

```
UpgradeNow force http://192.168.100.150/firmware/
```

- The "`list`" and "`force`" arguments cannot be used together.
- Once a firmware upgrade has begun, refrain from issuing reader commands, and do not allow the reader to lose power during the process.
- At some point during a firmware upgrade, all reader services and tag-reading activity halts. Network connections are interrupted, and the reader's heartbeat ceases until the upgrade is complete. **If you see the message, "WARNING: DO NOT UNPLUG THE POWER!", you must wait until the reader becomes responsive again – TCP connections are accepted, Alien> prompt on the serial line, or the resumption of reader heartbeats. Resist the urge to pull the power plug! Interrupting power at this time may result in a corrupted firmware load, and the reader will have to be returned to the factory.**

UpgradeNow Examples	
Command Response Command Response	// UpgradeNow using a preset UpgradeAddress (same firmware version) >UpgradeAddress = http://192.168.1.123/fw/alien UpgradeAddress = http://192.168.1.123/fw/alien >UpgradeNow ##### Running Network Upgrade 04/14/08 13:29:45 GMT+8A Upgrade URL : http://192.168.1.123/fw/alien Reader Version : 08041100 Package Downgrading : enabled List of Packages : ======================================= ALRx800_fw_080411.tar.aef ======================================= control line : ALRx800_fw_080411.tar.aef package version : 08041100 ...skipping ALRx800_fw_080411.tar.aef (same version) ##### No upgrades performed

Command
Response
	// UpgradeNow, with explicit address (new firmware version)
	>UpgradeNow http://192.168.1.123/fw/alien
	##### Running Network Upgrade 08/13/08 10:44:55 GMT+7A
	Upgrade URL : http://192.168.1.123/fw/alien
	Reader Version : 08081200
	Package Downgrading : enabled
	List of Packages :
	=======================================
	ALRx800_fw_080813_BETA.tar.aef
	=======================================
	control line : ALRx800_fw_080813_BETA.tar.aef
	package version : 08081300
	...downloading ALRx800_fw_080813_BETA.tar.aef
	...decrypting ALRx800_fw_080813_BETA.tar.aef
	...unpacking ALRx800_fw_080813_BETA.tar
	...installing ALRx800_fw_080813_BETA
	WARNING: DO NOT UNPLUG THE POWER!
	
	// The following output is visible only on the serial port:
	Alien>
	Alien Technology Corporation RFID Reader
	[ALIEN DRV] ALIEN READER driver ver 1.54.080111 UNLOADED
	[ALIEN DRV] ALIEN READER driver ver 1.54.080111 LOADED
	
	HOST activated (build: Aug 13 2008 02:10:19)
	
	Boot> Ready

UpgradeNow List & Force Examples	
Command Response	// UpgradeNow, listing the files >UpgradeNow list ALRx800_fw_080813_BETA.tar.aef

Command
Response
	// UpgradeNow with the force option (reload same firmware version)
	>UpgradeNow force
	##### Running Network Upgrade 08/13/08 10:52:52 GMT+7A
	Upgrade URL : http://10.10.82.10/dailybuild/
	Reader Version : 08081300
	Package Downgrading : enabled
	List of Packages :
	==========
	ALRx800_fw_080813_BETA.tar.aef
	==========
	control line : ALRx800_fw_080813_BETA.tar.aef
	package version : 08081300
	...downloading ALRx800_fw_080813_BETA.tar.aef
	...decrypting ALRx800_fw_080813_BETA.tar.aef
	...unpacking ALRx800_fw_080813_BETA.tar
	...installing ALRx800_fw_080813_BETA
	WARNING: DO NOT UNPLUG THE POWER!
	
	// The following output is visible only on the serial port:
	Alien>
	Alien Technology Corporation RFID Reader
	[ALIEN DRV] ALIEN READER driver ver 1.54.080111 UNLOADED
	[ALIEN DRV] ALIEN READER driver ver 1.54.080111 LOADED
	
	HOST activated (build: Aug 13 2008 02:10:19)
	
	Boot> Ready

NetworkTimeout

9900 | 9680 | 9650

When the reader receives a command on its Command port, it opens a TCP socket and waits for data to arrive. If inbound communication ceases, rather than hold the socket open indefinitely the reader waits a period of time then automatically closes the connection, ignoring any partial command it may have already received. This time period is the NetworkTimeout.

- Allowed Values: Integer (10..65535)
- Default Value: 90 seconds

NetworkTimeout Examples	
Command Response	>NetworkTimeout? NetworkTimeout = 90
Command Response	>NetworkTimeout = 120 NetworkTimeout = 120

	// Example Behavior (using Telnet)
	>NetworkTimeout = 15
	NetworkTimeout = 15
	
	// Wait more than 15 seconds
	Connection Timeout.
	Closing Connection…Bye!
	
	// Telnet session ends

CommandPort

9900 | 9680 | 9650

The reader can be configured and operated over the network using standard network sockets. The CommandPort settings are used to assign and retrieve the exact port number used by the reader for this network connectivity.

- Allowed Values: Integer(1..65535)
- Default Value: 23 (the standard Telnet port)
- Changes to this setting do not affect serial communication and/or Web communication with the reader.
- Changes to this setting don't take affect until the reader reboots.

CommandPort Examples	
Command	>CommandPort?
Response	CommandPort = 23
Command	>CommandPort = 10004
Response	CommandPort = 10004

CommandPortLocal

9900 | 9680 | 9650

The `CommandPortLocal` command is used to assign the port number for TCP connections to the reader originated from the reader itself (from the *localhost*). The default value for this port is 2300. This feature is intended to support (Ruby) on-reader applications. It is an independent command channel in addition to the standard command channel on port 23 (assigned by the `CommandPort` command). Using port 2300 leaves the standard port 23 available for external connections to the reader, so reader can be controlled from both the on-reader application and the command channel on port 23 at the same time. This could be desirable or not depending on the logic of your application, so select whichever port works for your use case. To disable the external command channel on port 23 set `AcceptConnections= local`.

- Allowed Values: Integer(1..65535)
- Default Value: 2300
- Changes to this setting do not affect serial communication and/or Web communication with the reader.
- Changes to this setting don't take affect until the reader reboots.

CommandPortLocal Examples	
Command	>CommandPortLocal?
Response	CommandPort = 2300
Command	>CommandPortLocal = 2345
Response	CommandPortLocal = 2345

AcceptConnections

9900 | 9680 | 9650

The reader can be restricted to only accept TCP connections originated from the reader itself (from the *localhost*). The command `AcceptConnections = {ANY | LOCAL}` controls this behavior.

Setting `AcceptConnections = LOCAL` prevents any TCP connections, initiated from outside the reader, from being accepted. Only connections originated from within the reader (for example from custom Ruby user applications running on the reader) will be accepted.

Changes must be Saved and the reader rebooted before the new behavior takes effect.

- Allowed Values: ANY | LOCAL
- Default Value: ANY
- Changes to this setting do not affect serial communication and/or Web communication with the reader.
- Changes to this setting don't take affect until the reader reboots.

AcceptConnections Examples	
Command Response	>AcceptConnections? AcceptConnections = ANY
Command Response	> AcceptConnections = local AcceptConnections = local

Ping

9900 | 9680 | 9650

The ping command is used for testing network connectivity between the reader and a specified network address. The traditional form of the ping command requires only the IP address (or hostname) of the target device. The reader sends an ICMP "ECHO_REQUEST" command to the target, which immediately replies to the reader if it successfully receives the request. The ping command not only tests for a successful connection, but also indicates the round-trip time for the transaction, which can be a useful indication of latencies present on the network.

Alien readers also provide a modified version of the ping command, where you specify a port number to connect to, in addition to the target's address. In this case, the reader attempts to open a socket to the device on that port number. It then reports not only a successful network route to the target, but also whether the target is listening for connections on the specified port. You can therefore test to see if your MailServer or NotifyAddress properties were set correctly.

Ping (IP Address) Examples	
Command Response	// Successful Ping: >ping 192.167.1.200 PING 192.167.1.200 (192.167.1.200): 56 data bytes 64 bytes from 192.167.1.200: icmp_seq=0 ttl=128 time=1.6 ms 64 bytes from 192.167.1.200: icmp_seq=1 ttl=128 time=1.2 ms 64 bytes from 192.167.1.200: icmp_seq=2 ttl=128 time=1.2 ms --- 192.167.1.200 ping statistics --- 3 packets transmitted, 3 packets received, 0% packet loss round-trip min/avg/max = 1.2/1.3/1.6 ms
Command Response	// Unsuccessful Ping: >ping 192.168.1.199 PING 192.168.1.199 (192.168.1.199): 56 data bytes --- 192.168.1.199 ping statistics --- 3 packets transmitted, 0 packets received, 100% packet loss

Ping (IP Address : Port) Examples	
Command Response	// Successful Ping: >ping 192.167.1.200:3600 Opening Socket at 192.168.1.200:3600 [192.168.1.200:3600] Remote Host Located Establishing Connection Established Connection Closing Connection Closed Connection Socket Successfully Located, Opened and Closed
Command Response	// Unsuccessful Ping: >ping 192.168.1.199:3600 Opening Socket at 192.168.1.199:3600 [192.168.1.199:3600] Remote Host Located Establishing Connection Error: Unable to Open Socket on Remote Host (Timeout)

HeartbeatPort

9900 | 9680 | 9650

The reader can be configured to periodically send out a heartbeat message to the network. This heartbeat takes the form of a single UDP packet (Universal Datagram Packet) broadcast out to the entire subnet or a particular address.

The HeartbeatPort command allows you to configure the actual port number that this packet is sent out to.

Listening for this heartbeat can be used to initially locate a reader on a network and subsequently make sure that the reader is still alive.

- Allowed Values: Integer(0..65535)
- Default Value: 3988

See chapter 2 for more information about the reader heartbeat.

HeartbeatPort Examples	
Command Response	>HeartbeatPort? HeartbeatPort = 3004
Command Response	>HeartbeatPort=10002 HeartbeatPort = 10002

HeartbeatTime

9900 | 9680 | 9650

The reader can be configured to periodically send out a heartbeat message to the network. This heartbeat takes the form of a single UDP packet (Universal Datagram Packet) broadcast out to the entire subnet or a particular address.

The time interval between heartbeats can be assigned and retrieved using this command.

- Allowed Values: Integer(0..86400)
- Default Value: 30
- All intervals are specified in seconds.
- A setting of zero (seconds) suspends the output of any further heartbeats.

HeartbeatTime Examples	
Command	>HeartbeatTime?
Response	HeartbeatTime = 30
Command	>HeartbeatTime=60
Response	HeartbeatTime = 60

HeartbeatAddress

9900 | 9680 | 9650

The reader can be configured to periodically send out a heartbeat message to the network. This heartbeat takes the form of a single UDP packet (Universal Datagram Packet) broadcast out to the entire subnet or a particular address.

The address of the host to receive these packets is defined by the HeartbeatAddress command.

- The default value 255.255.255.255 is a special "multicast" address, which enables all devices on the subnet to receive the packets.

HeartbeatAddress Examples	
Command	>HeartbeatAddress?
Response	HeartbeatAddress = 255.255.255.255
Command	>HeartbeatAddress =10.1.70.17
Response	HeartbeatAddress = 10.1.70.17

HeartbeatCount

9900 | 9680 | 9650

The HeartbeatCount property specifies how many heartbeat messages the reader broadcasts after it boots. After this point the reader stops sending heartbeat messages until the HeartbeatCount is changed, or the reader is rebooted.

- Allowed Values: Integer(-1..65535)
- Default Value: -1 (the reader sends heartbeat messages indefinitely)

HeartbeatAddress Examples	
Command	>HeartbeatCount?
Response	HeartbeatCount = -1
Command	>HeartbeatCount = 5
Response	HeartbeatCount = 5

HeartbeatNow

9900 | 9680 | 9650

The HeartbeatNow command forces the reader to immediately issue a UDP heartbeat, regardless of the HeartbeatTime or HeartbeatCount values. The command returns the entire XML-formatted heartbeat packet that was broadcast.

HeartbeatNow Example	
Command Response	>HeartbeatNow <?xml version="1.0" encoding="UTF-8"?> <Alien-RFID-Reader-Heartbeat> <ReaderName>Bob's 9800</ReaderName> <ReaderType>Alien RFID Tag Reader, Model: ALR-9800 (Four Antenna / Multi-Protocol / 902-928 MHz)</ReaderType> <IPAddress>192.168.100.150</IPAddress> <CommandPort>23</CommandPort> <HeartbeatTime>3</HeartbeatTime> <MACAddress>00:80:66:10:11:6A</MACAddress> <ReaderVersion>08.06.02.00</ReaderVersion> </Alien-RFID-Reader-Heartbeat>

WWWPort

9900 | 9680 | 9650

The reader's web server normally listens on port 80 for incoming connections (the default www port). The WWWPort can be changed to any allowed port number, although care must be taken not to use port number that is commonly used for other services, such as telnet, NTP, SNMP, etc.

- Allowed Values: Integer(0..65535)
- Default Value: 80
- Setting WWWPort to 0 effectively disables the reader's web server, which may be desired by some for security reasons. Keep in mind, though, that one of the main mechanisms for upgrading the reader's firmware uses the web server.
- Changes to WWWPort take effect immediately.

WWWPort Examples	
Command Response	>WWWPort? WWWPort = 80
Command Response	>WWWPort = 8080 WWWPort = 8080

HostLog

9900 | 9680 | 9650

The HostLog dumps a log of all the host activity (connects, disconnects, timeouts, etc.). This can be used to figure out when connections were made to the reader, whether serial or TCP, and the IP address of the connecting host (if TCP).

- HostLog is a "get" command – "Get HostLog" or "HostLog?".
- HostLog data is reset each time the reader boots, and over the course of operations the associated log files are periodically rotated out when they reach a certain size. This causes older log entries to be lost.

Following is an annotated example of a HostLog:

HostLog Examples	

Command
Response
	>HostLog?
	// Shows initial startup, and automatic "accept" of serial interface
	Oct 4 16:18:00 TCP: Initializing RFID Server on Port: 23
	Oct 4 16:18:00 SER: Accepted serial connection
	
	// A user at 10.10.82.75 is connected
	Oct 4 16:18:36 TCP: REQUEST connection from 10.10.82.75 port 1366
	Oct 4 16:18:36 TCP: ACCEPTED connection from 10.10.82.75 port 1366
	Oct 4 16:18:36 TCP: There are 1 clients active
	
	// User at 10.10.82.75 allowed his connection to timeout:
	Oct 4 16:20:45 TCP: No data within 90 seconds! Closing host socket...
	Oct 4 16:20:45 TCP: CLOSING connection from 10.10.82.75.
	
	// A user at 10.10.82.50 connects, but fails to enter the correct password
	Oct 4 16:20:59 TCP: REQUEST connection from 10.10.82.50 port 4358
	Oct 4 16:20:59 TCP: ACCEPTED connection from 10.10.82.50 port 4358
	Oct 4 16:20:59 TCP: There are 1 clients active
	Oct 4 16:21:00 TCP: Error: Login failed password
	Oct 4 16:21:00 TCP: Error: Invalid Username and/or Password
	Oct 4 16:21:01 TCP: CLOSING connection from 10.10.82.50.
	
	// The user at 10.10.82.75 connects again:
	Oct 4 17:02:49 TCP: REQUEST connection from 10.10.82.75 port 4506
	Oct 4 17:02:49 TCP: ACCEPTED connection from 10.10.82.75 port 4506
	Oct 4 17:02:49 TCP: There are 1 clients active
	
	// 10.10.82.50 tries to reconnect but refused while 10.10.82.75 is active:
	Oct 4 17:02:58 TCP: REQUEST connection from 10.10.82.50 port 1127
	Oct 4 17:02:58 TCP: ConnectionCount=1, Child PID=26530.
	Oct 4 17:02:58 TCP: Connection count limit exceeded. Refusing new connection
	
	// Finally, the user at 10.10.82.75 gracefully closes his connection
	Oct 4 17:05:15 TCP: CLOSING connection from 10.10.82.75.

DebugHost

9900 | 9680 | 9650

The DebugHost command can be used to log each and every command that is issued to the reader, from both the serial and TCP interfaces. The list of commands is viewed with the HostLog command (see above).

It is not a good idea to leave DebugHost on, unless you are diagnosing a specific problem, e.g. "why is my PersistTime always being reset?" Logging each and every command introduces some processing overhead, and since there is limited space available in the log file, filling it with extraneous data pushes older entries off the log.

- Allowed Values: "ON" | "OFF"
- Default Value: "OFF"

Following is an example of using the DebugHost command, and an annotated example the HostLog, after DebugHost has been on for a time period:

DebugHost Examples	
Command	>DebugHost = On
Response	DebugHost = ON
Command	>get HostLog

Response	// Some commands were issued on the serial interface:
	Oct 5 08:41:51 SER: [readername = This is my reader now!]
	Oct 5 08:42:31 SER: [tagtype=16]
	Oct 5 08:42:32 SER: [t]
	Oct 5 08:42:33 SER: [t]
	Oct 5 08:42:34 SER: [t]
	Oct 5 08:42:34 SER: [t]
	
	// A TCP connection is made, and they issue some commands:
	Oct 5 08:42:42 TCP: REQUEST connection from 10.10.82.50 port 1179
	Oct 5 08:42:42 TCP: ACCEPTED connection from 10.10.82.50 port 1179
	Oct 5 08:42:42 TCP: There are 1 clients active
	Oct 5 08:42:54 TCP: [get readername]
	Oct 5 08:42:56 TCP: [q]
	Oct 5 08:42:56 TCP: CLOSING connection from 10.10.82.50.
	
	// Finally, the command that initially generated this dump:
	Oct 5 08:43:00 SER: [hostlog?]

Time Commands

The time at which tags are read by a reader is particularly important for many applications. For this reason, the reader has three time commands to ensure that the onboard real-time clock is always set accurately.

TimeServer

9900 | 9680 | 9650

The reader can synchronize with a network time service to accurately set its internal clock every time it is rebooted, and keep it adjusted throughout operation. The protocol it uses is called the Network Time Protocol (NTP), which typically returns the time in UTC format.

In order to use this feature, a TimeServer must be specified. This is the network address of a machine that is constantly running the NTP service. In the US there are a number of machines owned and operated by the government, explicitly providing the time and date to Internet users.

- By default the reader is configured to connect to one of these machines on boot-up to get the current time. After boot-up, the reader continually adjusts its clock to match the current time reported by the NTP service.
- For a more in-depth description of this server, and a list of other publicly accessible Daytime Protocol Servers, see: http://www.boulder.nist.gov/timefreq/service/its.htm
- The default setting for this command is one of 132.163.4.101-103, the primary NIST network time servers. Some alternative time servers are:

  time-a.nist.gov : 129.6.15.28
  time-b.nist.gov : 129.6.15.29
  time.nist.gov : 192.43.244.18

- After making changes with this command, you must save and reboot the reader to implement the changes.
- If the reader is set up to use DHCP for network configuration, the DHCP server can supply a TimeServer to use, which overrides any specific TimeServer you may set up with this command.
- The reader can accept a list of TimeServers to use, either as provided by a DHCP server, or by giving a space-separated list of servers in the TimeServer command. This provides redundancy and improved time precision.

TimeServer Examples	
Command Response	>TimeServer? TimeServer = 129.6.15.28
Command Response	>TimeServer = 129.6.15.28 TimeServer = 129.6.15.28
Command Response	// Setting & Getting multiple NTP servers >TimeServer = 132.163.4.101 132.163.4.102 TimeServer = 132.163.4.101 132.163.4.102

TimeZone

9900 | 9680 | 9650

These commands allow the current time zone to be assigned to or retrieved from the reader. The time zone specifies the number of hours that must be added to or subtracted from UTC (Coordinated Universal Time; also known as GMT or Zulu) to determine a local time reference.

For example, to convert from UTC to Pacific Standard Time, set the TimeZone to –8. To convert from UTC to Pacific Daylight Time, set the TimeZone to –7.

- Allowed Values: Integer(-12..12)
- Default Value: -7 (Pacific Daylight Time)
- PDT is -7 because PDT is UTC time *minus 7 hours*.
- For more information about time zones, servers and UTC, refer to the Website listed under the Get/Set TimeServer command.

The TimeZone parameter is only useful if the TimeServer is used to automatically set the system clock. In this case, the TimeServer always retrieves the time in UTC format and needs to be offset to reflect local time using this parameter.

The reader does not automatically adjust for Daylight Saving Time. A convenient way to adjust for DST is to manually advance or retard the TimeZone value by one hour.

TimeZone Examples	
Command	>TimeZone?
Response	TimeZone = -8
Command	>TimeZone = 3
Response	TimeZone = 3

Time

9900 | 9680 | 9650

These commands allow the current time to be assigned to or retrieved from the reader.

- Times used by this command are always specified in local time, as defined by the TimeZone command.
- Times are always specified by the format YYYY/MM/DD hh:mm:ss.
- Changes made with this command take effect immediately.

Time Examples	
Command	>Time?
Response	Time = 2002/6/3 9:23:01
Command	>Time = 2002/6/3 19:23:01
Response	Time = 2002/6/3 19:23:01

External I/O Commands

These commands allow you to configure and retrieve data from the reader's external input/outputs. The concept of a TagList for buffering tag reads has been extended to a new IOList structure and associated commands, for buffering and reporting changes to the external inputs and outputs, which can be retrieved at a later time. There is also an IOStream mechanism that streams I/O events to a remote listener over TCP/IP, so that host applications do not need to constantly poll the reader to determine when an I/O changes state.

ExternalInput

9900 | 9680 | 9650

The reader monitors its external inputs, which can subsequently be controlled by external proximity detectors and other input devices such as "electric eyes" and magnetic switches. This command allows these external input values to be obtained. Please refer to the Hardware Setup Guide for pin-out diagrams.

- The command returns a single byte result that represents the bitmap of the external input states. Bit 0 (LSB) represents the state of input #0, bit 1 represents the state of input #1, etc.

ExternalInput Examples	
Command	`>ExternalInput?`
Response	`ExternalInput = 2  (i.e., binary "10")`

ExternalOutput

9900 | 9680 | 9650

The reader controls its external outputs, which can subsequently be used to control external devices such a doors/gates, security lights, etc. Please refer to the Hardware Setup Guide for pin-out diagrams.

With this command you can get or set the external output states. The single parameter/return value is an integer bitmap representing the state of the external outputs.

- Bit 0 represents the state of output #1, bit 1 represents the state of output #2, etc.
- For example, to set output #2 high and output #1 low, use the bitmap of $2_{decimal}$, which translates to 10_{binary}.

ExternalOutput Examples	
Command	`>ExternalOutput = 2`
Response	`ExternalOutput = 2`
Command	`>ExternalOutput?`
Response	`ExternalOutput = 2`

InvertExternalInput

9900 | 9680 | 9650

In the default configuration, a high voltage applied to an external input results in an external input reading of "1", or "on". Depending on the electronics of the connected device, a high voltage applied to the pin may actually be the result of the device being "inactivate" instead of "activate". To accommodate this case and avoid possible confusion when examining bitmaps of external input states, turning on InvertExternalInput reverses the "sense" of all the external inputs, effectively resulting an external input value to be "1" when the pin voltage is driven low, and vice versa..

- Allowed Values: "ON" | "OFF"
- Default Value: "OFF"

InvertExternalInput Examples	
Command Response	>ExternalInput? ExternalInput = 0
Command Response	>InvertExternalInput = ON InvertExternalInput = ON
Command Response	>InvertExternalInput? InvertExternalInput = ON
Command Response	>ExternalInput? ExternalInput = 15 (inverted value - inputs haven't changed)

InvertExternalOutput

9900 | 9680 | 9650

In the default configuration, setting an external output to "1", or "on", causes the voltage on that output pin to go high. Depending on the electronics of the connected device, a high voltage on the pin may cause that device to "activate" or vice versa. To accommodate this case and avoid possible confusion when specifying bitmaps of desired external output states, turning on InvertExternalOutput reverses the "sense" of all the external outputs, effectively causing to pin voltage to go high when the external output is set to "0", and vice versa.

- Allowed Values: "ON" | "OFF"
- Default Value: "OFF"
- When ON, the inversion of output states is applied both in setting and getting the output states. It is therefore a transparent change, unless you test the voltage on the output pins.
- The state of InvertExternalOutput has no effect on the InitExternalOutput during boot-up. The state defined by InitExternalOutput is applied to the outputs regardless of the InvertExternalOutput setting.

InvertExternalOutput Examples	
Command Response	>ExternalOutput? ExternalOutput = 0
Command Response	>InvertExternalOutput = ON InvertExternalOutput = ON
Command Response	>InvertExternalOutput? InvertExternalOutput = ON
Command Response	>ExternalOutput? ExternalOutput = 15 (inverted value - outputs haven't changed)

InitExternalOutput

9900 | 9680 | 9650

When the reader is powered up, it sets the external outputs to states defined by the InitExternalOutput attribute. This gives you the flexibility to choose which outputs should default to high, and which outputs should default to low – a very important consideration if there are mechanical devices controlled by the readers I/O port.

With this command you can get or set the initial external output states. The single parameter/return value is an integer bitmap representing the states of the external outputs during and after a startup.

- Bit 0 represents the desired state of output #1, bit 1 represents the desired state of output #2, etc.
- For example, to set output #2 high and output #1 low, use the bitmap of $2_{decimal}$, which translates to 0010_{binary}.
- Changes made with this command take effect during the next reboot.
- The bitmap that you specify is applied directly to the outputs upon startup, without any inversion that may have been set with InvertExternalOutput.

InitExternalOutput Examples	
Command	>InitExternalOutput = 7 *(0111$_{binary}$ = #1, 2, 3 high, #4 low)*
Response	InitExternalOutput = 7
Command	>InitExternalOutput?
Response	InitExternalOutput = 7

Get IOList (ios)

9900 | 9680 | 9650

You can retrieve the reader's stored IOList with the "Get IOList" command. The reader monitors every change to its digital inputs and outputs, and records them on an internal IOList. The I/O events are listed in order of their occurrence. When you retrieve the list, those stored events are cleared.

- The maximum number of I/O events that can be stored in the IOList is 100,000.
- "Get IOList" and "ios" can be used interchangeably.

The format of the data returned by this command is specified using the IOListFormat command, and you can specify whether to track inputs, outputs, or both (interleaved by event time, or listed separately), using the IOType command. Both of these commands are described later. In general, each entry on the IOList indicates whether the event was a change of an digital input (DI) or digital output (DO), a timestamp of when the event took place, and the new (bit mask) state of the inputs or outputs.

Note that the value indicated in the IOList doesn't tell you explicitly which I/O changed, or even in which direction – it simply indicates the new state of all the I/O. To determine which I/O changed state, and in which direction, you have to compare the new value with a previous value.

IOList Examples	
Command	>get IOList
Response	IO:DI, Time:2007/01/18 10:15:57.018, Data:0 IO:DO, Time:2007/01/18 10:16:07.486, Data:0
Command	>ExternalOutput = 15
Response	ExternalOutput = 15
Command	>ios
Response	IO:DO, Time:2007/01/18 11:43:42.582, Data:15

IOPersistTime

9900 | 9680 | 9650

The IOPersistTime specifies the length of time an IO event remains in the reader's internal IOList. You can turn off the IOList completely, hold on to IO events until you later ask for them, with an optional "persist time" after which stale events will be removed from the list.

- Allowed Values: Integer(-1..86400)
- Default Value: 0 (no IO events are stored)
- Persist times are specified in seconds.
- Setting the IOPersistTime to -1 causes the history to build indefinitely until a `get IOList` command is issued; at this point the IOList is returned, and then immediately cleared.
- An IOPersistTime > 0 means the IO events fill remain on the IOList for that amount of time before they will be automatically removed. If you don't ask for the IOList before then, those events are lost.
- IO events are always streamed first (if IOStreamMode is configured) before they are stored on the IOList. The IOPersistTime has no effect on IO streaming.

IOPersistTime Examples	
Command Response	>IOPersistTime? IOPersistTime = 0
Command Response	>IOPersistTime=300 IOPersistTime = 300

IOType

9900 | 9680 | 9650

You specify which I/O events you are interested in tracking and reporting with the IOType command. The IOType value is specified with one of the logical enumerations, DI, DO, DIO, as follows:

DI - inputs only
DO - outputs only
DIO - inputs and outputs, interleaved by event time (default value)

IOType enumerations can be combined with the "|" (vertical bar), space, comma, semi-colon (;), or ampersand (&) characters. The reader always reformats them to be separated by a comma.
The default value of `IOType` is `DI`.

At present, the only ones that make sense to combine are DI and DO, but in the future there may addition I/O events that you can combine. For example: `IOType = DI, DO`
Note, '`DI,DO`' is different from '`DIO`' as the inputs are listed separately from outputs (not interleaved)

IOType Examples	
Command Response	>IOType? IOType = DIO
Command Response	>get IOList IO:DI, Time:2007/01/19 09:48:16.471, Data:0 IO:DO, Time:2007/01/19 09:48:26.918, Data:0 IO:DI, Time:2007/01/19 10:08:17.627, Data:13 IO:DI, Time:2007/01/19 10:08:32.718, Data:9 IO:DO, Time:2007/01/19 10:08:35.266, Data:11 IO:DO, Time:2007/01/19 10:08:37.856, Data:3 IO:DI, Time:2007/01/19 10:08:39.766, Data:13 IO:DI, Time:2007/01/19 10:08:40.538, Data:15 IO:DO, Time:2007/01/19 10:08:42.867, Data:0 IO:DO, Time:2007/01/19 10:08:45.477, Data:211
Command Response	*// Assuming the same I/O events as above…* >IOType = DI IOType = DI >ios IO:DI, Time:2007/01/19 09:48:16.471, Data:0 IO:DI, Time:2007/01/19 10:08:17.627, Data:13 IO:DI, Time:2007/01/19 10:08:32.718, Data:9 IO:DI, Time:2007/01/19 10:08:39.766, Data:13 IO:DI, Time:2007/01/19 10:08:40.538, Data:15
Command Response	*// Assuming the same I/O events…* >IOType = DO IOType = DO >ios IO:DO, Time:2007/01/19 09:48:26.918, Data:0 IO:DO, Time:2007/01/19 10:08:35.266, Data:11 IO:DO, Time:2007/01/19 10:08:37.856, Data:3 IO:DO, Time:2007/01/19 10:08:42.867, Data:0 IO:DO, Time:2007/01/19 10:08:45.477, Data:211
Command Response Command Response	*// Assuming the same I/O events…* >IOType = DI, DO IOType = DI,DO IO:DI, Time:2007/01/19 09:48:16.471, Data:0 IO:DI, Time:2007/01/19 10:08:17.627, Data:13 IO:DI, Time:2007/01/19 10:08:32.718, Data:9 IO:DI, Time:2007/01/19 10:08:39.766, Data:13 IO:DI, Time:2007/01/19 10:08:40.538, Data:15 IO:DO, Time:2007/01/19 09:48:26.918, Data:0 IO:DO, Time:2007/01/19 10:08:35.266, Data:11 IO:DO, Time:2007/01/19 10:08:37.856, Data:3 IO:DO, Time:2007/01/19 10:08:45.477, Data:211

IOListFormat

9900 | 9680 | 9650

The IOListFormat command lets you specify the formatting of IOLists. The command takes a text string as its argument, and can be one of the following:

Format	Description
Text	IOList displayed as plain text messages, one I/O event per line.
Terse	IOList displayed as plain text messages, one I/O event per line, but with a simplified output showing just the type of event, timestamp, and value, with no labels.
XML	IOList displayed in XML text format
Custom	IOList displayed in the format given by IOListCustomFormat.

- All timestamps on the IOList include millisecond resolution, either with .xxx after the seconds in the human-readable form, or as the number of milliseconds since the UNIX epoch (Jan 1, 1970).
- The default value is "Text".
- Text-formatted IOLists take the general form:

```
IO:{DI|DO}, Time:YYYY/MM/DD hh:mm:ss.xxx, Data:<integer value>
```

For example:

```
IO:DI, Time:2007/01/19 09:48:16.471, Data:0
IO:DO, Time:2007/01/19 09:48:26.918, Data:0
```

- Terse-formatted IOLists take the general form (1 = DI, 2 = DO):

```
{1|2},<millisecond timestamp>,<integer value>
```

For example:

```
2,1169231710479,1  // DOs went to state 1
1,1169231712696,13 // DIs went to state 1
```

- XML-formatted IOLists take the general form:

```
<?xml version="1.0" encoding="UTF-8"?>
<Alien-RFID-IO-List>
   <Alien-RFID-IO>
      <Type>{DI | DO}</Type>
      <Time>YYYY/MM/DD hh:mm:ss.xxx</Time>
      <Data><integer value></Data>
   </Alien-RFID-IO>
   …
</Alien-RFID-IO-List>
```

For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<Alien-RFID-IO-List>
   <Alien-RFID-IO>
      <Type>DO</Type>
      <Time>2007/01/19 10:39:16.349</Time>
      <Data>5</Data>
   </Alien-RFID-IO>
</Alien-RFID-IO-List>
```

- Custom-formatted IOLists use the formatting string specified by IOListCustomFormat, substituting actual values in place of defined "custom tokens". See the IOListCustomFormat section (below) for more information and examples.

IOListFormat Examples	
Command Response	>IOListFormat? IOListFormat = Text
Command Response	>IOListFormat = XML IOListFormat = XML

IOListCustomFormat

9900 | 9680 | 9650

The IOListCustomFormat command allows a customized IOList to be defined. Once the format has been defined it can be applied by issuing the command "`IOListFormat = Custom`".

The IOListCustomFormat command takes a single text line argument that defines how each I/O event should be represented. This argument can be made up of a mixture of text and custom tokens, as defined in the table below. The maximum length of a IOListCustomFormat definition is 255 characters.

When the reader is outputs a custom IOList, the tokens in the custom format are replaced with the actual values for each I/O event. You can use the short and long tokens interchangeably.

Short Token	Long Token	Description
%E	**${EVENT}**	The I/O event type (textual): "DI" or "DO"
%e	**${ID}**	The I/O event type (numeric): 1=DI, 2=DO
%T	**${TIME}**	Time of the I/O event (textual): hh:mm:ss.xxx
%t	**${MSEC}**	Time of the I/O event (numeric): msec since 01/01/1970
%d	**${DATE}**	Date of the I/O event: YY/MM/DD
%D	**${DATELONG}**	Date & Time of the I/O event: YY/MM/DD hh:mm:ss.xxx
%v	**${DATA}**	The new (bit mask) state of the DIs or DOs
%N	**${NAME}**	The reader's ReaderName
%H	**${HOST}**	The reader's Hostname
%I	**${IP}**	The reader's IP address
%M	**${MAC}**	The reader's MAC address

IOListCustomFormat Examples	
Command Response	>IOListCustomFormat = The new %E value is %v. IOListCustomFormat = The new %E value is %v. >ios The new DO value is 6. The new DI value is 3.
Command Response	>IOListCustomFormat = E=%E e=%e T=%T t=%t d=%d D=%D v=%v IOListCustomFormat = E=%E e=%e T=%T t=%t d=%d D=%D v=%v >IOList? E=DO e=2 T=14:04:48.286 t=1169244288286 d=2007/01/19 D=2007/01/19 14:04:48.286 v=2 E=DI e=1 T=14:04:51.732 t=1169244291732 d=2007/01/19 D=2007/01/19 14:04:51.732 v=13

Clear IOList

9900 | 9680 | 9650

The "Clear IOList" command instructs the reader to immediately clear its internal IOList.

Clear IOList Examples	
Command	>Clear IOList
Response	IO List has been cleared!

IOStreamMode

9900 | 9680 | 9650

The IOStreamMode command turns on or off the IOStream functionality. Rather than having to continually poll the reader to determine if a digital input value has changed, you can now configure the reader to stream digital I/O events to a TCP socket or the reader's serial port.

When IOStreamMode is on, each change to a digital input or digital output state causes the reader to stream the event information to the IOStreamAddress (see below). Only those events given by the IOType command (DI, DO, DIO, etc.) are streamed.

The format of the streamed data is specified by the IOStreamFormat and IOStreamCustomFormat commands (below).

- Allowed Values: "ON" | "OFF"
- Default Value: "OFF"
- If the reader is unable to successfully stream the data (TCP socket error, for instance), the data is lost.

IOStreamMode Examples	
Command	>IOStreamMode?
Response	IOStreamMode = OFF
Command	>IOStreamMode = on
Response	IOStreamMode = ON

IOStreamAddress

9900 | 9680 | 9650

The IOStreamAddress command specifies where IOStream event data should be sent. There are two delivery methods supported, as shown in the table below:

IOStreamAddress	Description
hostname:port	Send a message to a specified port on a networked machine. The address takes the form “hostname:port.” For example, “123.01.02.98:3450” or “listener.alientechnology.com:10002”
SERIAL	Send a message to the serial port. The word “serial” is used as the address. The word is not case sensitive.

- Note: care must be taken when streaming data to the serial port. I/O events can happen very fast, and may have a deleterious effect on the responsiveness of the serial port when issuing commands at the same time.

IOStreamAddress Examples	
Command Response	>IOStreamAddress? IOStreamAddress = 10.1.0.12:4000
Command Response	>IOStreamAddress = serial IOStreamAddress = SERIAL

IOStreamFormat

9900 | 9680 | 9650

The IOStreamFormat command lets you specify the formatting of IOStream data (separate from the IOList). The command takes a text string as its argument, and can be one of the following:

Format	Description
Text	IOStream data is displayed as plain text messages, one I/O event per line.
Terse	IOStream data is displayed as plain text messages, one I/O event per line, but with a simplified output showing just the type of event, timestamp, and value, with no labels.
XML	IOStream data is displayed in XML text format
Custom	IOStream data is displayed in the format given by IOStreamCustomFormat.

- All timestamps on the IOStream include millisecond resolution, either with .xxx after the seconds in the human-readable form, or as the number of milliseconds since the UNIX epoch (Jan 1, 1970).
- The default value is "Terse".
- The formatting of IOStream data is the same as for IOList data. See the description of IOListFormat for output formats and examples of each type.

IOStreamFormat Examples	
Command Response	>IOStreamFormat? IOStreamFormat = Terse
Command Response	>IOStreamFormat = Text IOStreamFormat = Text

IOStreamCustomFormat

9900 | 9680 | 9650

The IOStreamCustomFormat command allows a customized IOStream format to be defined. Once the format has been defined it can be applied by issuing the command "`IOStreamFormat = Custom`".

The IOStreamCustomFormat command takes a single text line argument that defines how each I/O event should be represented. This argument can be made up of a mixture of text and custom tokens, as defined in the table below. The maximum length of the IOStreamCustomFormat definition is 255 characters.

When the reader outputs an IOStream event, the tokens in the custom format are replaced with the actual values for the I/O event.

- The available custom tokens for the IOStream are the same as for IOList. See the IOListCustomFormat description for a table of allowed tokens.

- The default IOStreamCustomFormat is "%E,%v "

IOStreamCustomFormat Examples	
Command Response	>IOStreamCustomFormat? IOStreamCustomFormat = %E,%v
Command Response	>IOStreamCustomFormat = %E changed to %v at %T. IOStreamCustomFormat = %E changed to %v at %T.

IOStreamKeepAliveTime

9900 | 9680 | 9650

When the reader sends an IOStream event message out over the network, it needs to open a TCP socket. Opening and closing a socket entails some processor and network overhead, and if the messages are coming out of the reader rapidly, this overhead can become a burden that hinders reader responsiveness.

The IOStreamKeepAliveTime command sets the amount of time (in seconds) that the reader keeps its IOStream socket open. For rapid IOStream events, it may be advantageous for the reader to hold the socket open continuously. You do this by setting IOStreamKeepAliveTime to some value greater than the expected time between events.

On the other hand, holding the socket open places a burden on the network. For infrequent IOStream events, it is advantageous to leave the IOStreamKeepAliveTime small so that the socket is opened only long enough for a single message to be sent out, and is then closed automatically after message delivery.

- Allowed Values: Integer(0..32767)
- Default Value: 30 seconds
- The reader automatically reopens a socket that has timeout the next time it needs to send data.

IOStreamKeepAliveTime Examples	
Command Response	>IOStreamKeepAliveTime? IOStreamKeepAliveTime = 30
Command Response	>IOStreamKeepAliveTime = 2 IOStreamKeepAliveTime = 2

BlinkLED

9900 | 9680 | 9650

The incredibly useful BlinkLED command allows you to cycle the reader's LED status lights between two patterns with a specified duration for each state and the number of times to cycle. You might use this to signal an error condition or a directive to an operator nearby, or you might use it to verify that the status LEDs are all functioning.

```
BlinkLED = <LEDState1> <LEDState2> <duration> <count>
```

<LEDState1> and <LEDState2> are the two LED states, represented as decimal bitmasks with the following mappings. Bit 1 is the least-significant bit.

ALR-x800/9900	ALR-9680	ALR-9650
bit 1 – Ant 0	bit 1 – RF On	bit 1 – RF On
bit 2 – Ant 1	bit 2 – Read	bit 2 – Read
bit 3 – Ant 2	bit 3 – Fault (red)	bit 3 – Fault (red)
bit 4 – Ant 3	bit 4 – N/A	bit 4 – N/A
bit 5 – CPU	bit 5 – N/A	bit 5 – N/A
bit 6 – Read	bit 6 – N/A	bit 6 – N/A
bit 7 – Sniff	bit 7 – N/A	bit 7 – N/A
bit 8 – Fault (red)	bit 8 – N/A	bit 8 – N/A

<duration> is the time, in milliseconds, to hold each output state. <count> is the number of times to cycle through the two states.

- <LEDState> allowed values: Integer(0..255) (the useful maximum depends on how many LEDs the reader has)
- <duration> allowed values: Integer(0..2500)
- <count> allowed values: Integer(0..255)
- The total duration of the BlinkLED command (2 * <duration> * <count>) cannot exceed 25 seconds,
- No other commands may be issued while the BlinkLED command is executing. AutoMode is suspended until the BlinkLED command finishes. The reader does not read tags while BlinkLED is running.

BlinkLED Examples	
Command Response	// Flash all LEDs On and Off 3 times, pausing for 1 sec each time >BlinkLED = 0 255 1000 3 BlinkLED = 0 <--> 255 (1000 msec, 3 times)
Command Response	// Flash the fault light (ALR-x800/9900) rapidly for 2 sec total. >BlinkLED = 128 0 100 20 BlinkLED = 128 <--> 0 (100 msec, 20 times)

TagList Commands

TagList commands allow you to retrieve immediate listings of tags that have been read and saved by the reader, and to assign and retrieve TagList functional parameters.

Get TagList (t)

9900 | 9680 | 9650

You can retrieve the reader's stored TagList with the Get TagList command.

- The maximum number of tags that can be stored in the TagList is 6000. Some reader models also have the capability to retain some of the TagList data even over a power outage.
- "Get TagList" and "t" can be used interchangeably.

Using the Get TagList to retrieve the stored list only once:

- **If the reader is not in Autonomous Mode**, the reader immediately performs a full tag search (read and report) and displays its current internal TagList. The reply is a multi-line response, with each line listing an active tag. If the TagList is empty, the message "(No Tags)" is returned.
- **If the reader is in Autonomous Mode**, the reader just returns its current internal TagList.

The format of the data returned by this command is specified using the TagListFormat command, described below.

TagList Examples	
Command	`>get TagList`
Response	`Tag:8000 8004 0000 003B, Disc:2003/12/04 12:35:11, Last:2003/12/04 12:35:11, Count:3, Ant:0` `Tag:8000 8004 9999 0004, Disc:2003/12/04 12:35:11, Last:2003/12/04 12:35:11, Count:3, Ant:0`
Command	`>get TagList`
Response	`(No Tags)`

PersistTime

9900 | 9680 | 9650

The PersistTime specifies the length of time a tag's data remains in the reader's internal list of active tags.

- Allowed Values: Integer(-1..86400)
- Default Value: -1
- Persist times are specified in seconds.
- A zero persist time (0) guarantees that tags are not stored in the TagList. However issuing a `get TagList` command in Interactive Mode returns any tags immediately found even though they won't be stored in the TagList.
- Setting the persist time to -1 causes the history to build indefinitely until a `get TagList` command is issued; at this point the TagList is returned, and then immediately cleared. This is the default setting.

The maximum number of tags that can be stored in the internal TagList is 6,000. Once this tag limit is reached, older tag entries are replaced by newer ones. The ALR-9900+ has the additional feature of being able to periodically store acquired tag reads to non-volatile memory, so that in the event of power loss, those readers can reboot and come back with the same list of tag reads.

PersistTime Examples	
Command Response	>PersistTime? PersistTime = -1
Command Response	>PersistTime=300 PersistTime = 300

TagListFormat

9900 | 9680 | 9650

The Get and Set TagListFormat commands specify the formatting of TagLists. The command itself takes a text string as its argument, and can be one of the following:

Format	Description
Text	TagLists displayed as plain text messages, one tag ID per line.
Terse	TagLists displayed as plain text messages, one tag ID per line, but just TagID, Antenna, and ReadCount, with no labels.
XML	TagLists are displayed in XML text format
Custom	TagLists are displayed in the format described by TagListCustomFormat.

- Text-formatted TagLists take the following form:

```
Tag:E200 3411 B801 0108, Disc:2007/06/29 08:30:49, Last:2007/06/29 10:38:12,
Count:292, Ant:0, Proto:2
Tag:4461 7669 6445 2E4B, Disc:2007/06/29 10:38:13, Last:2007/06/29 10:38:13,
Count:187, Ant:1, Proto:2
```

- Terse-formatted TagLists take the following form:

```
1115 F268 81C3 C012,0,4
0100 0100 0002 0709,0,6
1054 A334 54E1 7409,0,2
```

  The fields given in the Terse format are: TagID, ReadCount, Antenna.

- XML-formatted TagLists take the form:

```
<?xml version="1.0" encoding="UTF-8"?>
<Alien-RFID-Tag-List>
   <Alien-RFID-Tag>
      <TagID>0000 0000 0000 0000 0000 0000</TagID>
      <DiscoveryTime>2005/05/31 17:39:13</DiscoveryTime>
      <LastSeenTime>2005/05/31 17:39:13</LastSeenTime>
      <Antenna>1</Antenna>
      <ReadCount>22</ReadCount>
      <Protocol>0</Protocol>
   </Alien-RFID-Tag>
   <Alien-RFID-Tag>
      <TagID>A5A5 FFFF 8000 8004 6546 6091</TagID>
      <DiscoveryTime>2005/05/31 17:39:13</DiscoveryTime>
      <LastSeenTime>2005/05/31 17:39:13</LastSeenTime>
      <Antenna>0</Antenna>
      <ReadCount>3</ReadCount>
      <Protocol>1</Protocol>
   </Alien-RFID-Tag>
   <Alien-RFID-Tag>
      <TagID>3000 2141 60C0 0400 0000 6013</TagID>
      <DiscoveryTime>2005/05/31 17:39:13</DiscoveryTime>
      <LastSeenTime>2005/05/31 17:39:14</LastSeenTime>
      <Antenna>1</Antenna>
      <ReadCount>19</ReadCount>
      <Protocol>2</Protocol>
   </Alien-RFID-Tag>
</Alien-RFID-Tag-List>
```

In all cases the following information is reported per tag:

- TagID: the tag's individual ID (EPC code).
- Disc: the time the tag was first read by the reader in the current session.
- Last: the most recent time the tag was read by the reader in the current session.
- Count/ReadCount: the number of times the tag has been read in the current session.
- Ant: the antenna port number where the tag was LAST seen. For multi-static readers, the antenna port reported is the transmit antenna.

TagListFormat Examples	
Command Response	>TagListFormat? TagListFormat = Text
Command Response	>TagListFormat = XML TagListFormat = XML

TagListCustomFormat

9900 | 9680 | 9650

The TagListCustomFormat command allows a customized TagList to be defined. Once the format has been defined it can be applied by issuing the command "`TagListFormat = Custom`".

The TagListCustomFormat command takes a single text line argument that defines how each tag should be represented on-screen. This argument can be made up of a mixture of text and tokens, as defined in the table below. The maximum length of a custom TagList format definition is 255 characters.

Custom tokens come in short and long forms. The traditional (short) tokens take the form %X, where "X" is a single character identifying the token, and the long tokens take the form ${var_name}. Short and long tokens may be used interchangeably.

When the RFID Reader is required to generate a TagList, the tokens and variables in the custom format are replaced with the actual values for each tag.

The following tokens apply to all Alien readers:

Short Tokens	Long Tokens	Description
%i	**${TAGIDW}**	Tag ID with a space between each pair of bytes, i.e., 8000 00FE 8010 2AB7
-	**${TAGIDB}**	Tag ID with a space between each byte, i.e., 80 00 00 FE 80 10 2A B7
%k	**${TAGID}**	Tag ID with no spaces at all, i.e., 800000FE80102AB7
%d	**${DATE1}**	Discovery date of tag, in format YY/MM/DD
%t	**${TIME1}**	Discovery time of tag, in format hh:mm:ss
-	**${MSEC1}**	Discovery date & time of tag, in milliseconds since 1/1/1970.
%D	**${DATE2}**	Last-seen date of tag, in format YY/MM/DD
%T	**${TIME2}**	Last-seen time of tag, in format hh:mm:ss
-	**${MSEC2}**	Last-seen date & time of tag, in milliseconds since 1/1/1970.
%r	**${COUNT}**	Read Count of tags, i.e., how many times the tag has been read
%a	**${TX}**	Antenna the tag was last seen at (for the ALR-x800, this is the transmit antenna)
%A	**${RX}**	(ALR-x800 only) Receive Antenna where the tag was last seen
%p	**${PROTO#}**	Integer value indicating the tag's protocol (0 = Class0, 1 = Class1/Gen1, 2 = Class2/Gen2)
%P	**${PROTO}**	String representation of the tag's protocol
%l	**${PCWORD}**	PC Word (Class1/Gen2)
%m	**${RSSI}**	Tag RSSI measurement
-	**${RSSI_MAX}**	The maximum RSSI value for this tag.
%N	**${NAME}**	The reader's ReaderName
%H	**${HOST}**	The reader's Hostname
%I	**${IP}**	The reader's IP address
%M	**${MAC}**	The reader's MAC address

The following tokens apply only to ALR-9000+ readers, since they expose data that is only available on enterprise-class readers:

Short Tokens	Long Tokens	Description
%s	**${SPEED}**	Tag speed (m/s) – impacts read performance
-	**${SPEED_MIN}**	The minimum (most negative) speed measured for this tag.
-	**${SPEED_MAX}**	The maximum (most positive) speed measured for this tag.
-	**${SPEED_TOP}**	The top speed (regardless of direction) for this tag.
-	**${DIR}**	Direction: "-" (approaching), "+" (receding), or "0" (stationary)
-	**${G2DATA1...4}**	Tag data corresponding to the 1st, 2nd, 3rd, or 4th AcqG2TagData entry
-	**${G2OPS}**	Results from all AcqG2Ops that were executed, entries separated by vertical bars (|).
-	**${G2OPS1…8}**	Results corresponding to 1 (of 8) of the numbered AcqG2Ops
-	**${AUTH}**	Tag's Alien dynamic authentication data (includes tag manufacturer's id code) Refer to the description of the `TagAuth` command for more details.
-	**${XPC}**	XPC Word(s) (Class1/Gen2)

Support for XPC Functionality

9900 | **9680** | **9650**

The information returned from EPC Gen2 tags, and from ISO18000-6c tags, includes a word (16 bits) of Protocol Control information (PC word). This information includes the length of the packet, and additional information about of the tag such as whether the User bank has information written to it, the ISO numbering system identifier, or the EPC information bits, including a bit for tagging hazardous materials. In addition, some Gen2 tags may have an additional word or two of information which are collectively referred to as the eXtended Protocol Control bits (XPC). These are defined for purposes such as identifying battery tags, indication of exceptional conditions at tags such as that they need attention (temperature too high for temperature tags, for example).

TagListCustomFormat Examples	
Command Response	>TagListCustomFormat = Here is a tag %i TagListFormat = Here is a tag %i
Command Response	>get TagList Here is a tag 8000 0000 0000 0808 Here is a tag 102F ED3D 0303 0001
Command Response	>TagListCustomFormat = Tag %k, read %r times from antenna %a TagListFormat = Tag %k, read %r times from antenna %a
Command Response	>get TagList Tag 8000000000000808, read 3 times from antenna 0 Tag 102FED3D03030001, read 120 times from antenna 1
Command Response	>TagListCustomFormat = Tag ${TAGIDW} was going ${SPEED} m/s. TagListFormat = Tag ${TAGIDW} was going ${SPEED} m/s.
Command Response	>t Tag 8000 0000 0000 0808 was going 1.064 m/s. Tag 102F ED3D 0303 0001 was going -0.003 m/s.

The information available in the PC and XPC bits is standardized the EPC and ISO standards committees. The anticipated uses and the meanings of the bits may change in the future, and other XPC words may be defined.

The software in Alien readers will decode and present the bits from any PC and XPC words in custom tag lists by using the ${XPC} and ${PCWORD} custom tag list format token.

For example:

```
Alien>TagListCustomFormat = id:%i pc:${PCWORD} xpc:${XPC}
TagListCustomFormat = id:%i pc:${PCWORD} xpc:${XPC}

Alien>TagListFormat = custom
TagListFormat = custom

Alien>t
id:E9C1 E9C2 E9C3 E9C4 E9C5 E9C6 pc:3A00 xpc:0002
```

Notice that the reported PC word has a value of **0x3A00** which corresponds to the EPC length of 7 words. Since this tag has an extra XPC word, it adds 1 to the EPC length of 6 words when reporting the PC word value.
If the EPC bank is read directly with the G2Read command the bits at the PC word location (second word of the EPC bank) have a value of **0x3200**. ('**2**' indicates that the XPC word is present):

```
Alien>g2read=1 0 0
G2Read = 14 1D 32 00 E9 C1 E9 C2 E9 C3 E9 C4 E9 C5 E9 C6
```

TagDataFormatGroupSize

9900 | 9680 | 9650

The new TagDataFormatGroupSize command is used to specify the grouping of bytes when formatting tag data. This parameter affects formatting of tag's EPC and AcqG2TagData values for Text, XML and terse formats. It does not affect the formatting of programming-related commands, such as ProgramEPC and G2Read.

The format is as follows:

```
TagDataFormatGroupSize = <group size>
```

where <group size> is the number of bytes in a group (0 to 2). The default value is 2.

This feature could be useful for users designing their own data parsers.

TagDataFormatGroupSize Examples	
Command Response	>TagDataFormatGroupSize? TagDataFormatGroupSize = 2
Command Response	>AcqG2TagData = 1 2 2 AcqG2TagData = 1 2 2
Command Response	>t Tag:E200 9002 5214 0283 1740 60EB, Disc:2010/02/22 16:47:31.272, Last:2010/02/22 16:47:31.272, Count:1, Ant:0, Proto:2, D1:**E200 9002**
Command Response	>TagDataFormatGroupSize = 1 TagDataFormatGroupSize = 1
Command Response	Alien>t Tag:E2 00 90 02 52 14 02 83 17 40 60 EB, Disc:2010/02/22 16:48:17.257, Last:2010/02/22 16:48:17.257, Count:1, Ant:0, Proto:2, D1:**E2 00 90 02**
Command Response	>TagDataFormatGroupSize = 0 TagDataFormatGroupSize = 0
Command Response	Alien>t Tag:E20090025214028317406 0EB, Disc:2010/02/22 16:44:52.878, Last:2010/02/22 16:44:52.878, Count:1, Ant:0, Proto:2, D1:**E2009002**

TagListAntennaCombine

9900 | 9680 | 9650

The TagListAntennaCombine command turns on or off the antenna combine mode. When TagListAntennaCombine is ON, the reader combines tag IDs into a single TagList even if the Ids are read by different antennas. Setting this value to OFF forces the TagList to keep multiple copies of a tag ID for each antenna where it is seen.

For example, reading a tag that is visible to both antenna 0 and antenna 1, with an AntennaSequence of "0, 1", would give the following TagList:

TagListAntennaCombine = ON

```
Tag:8000 8004 2665 8426, Count:2, Ant:1
```

TagListAntennaCombine = OFF

```
Tag:8000 8004 2665 8426, Count:1, Ant:0
Tag:8000 8004 2665 8426, Count:1, Ant:1
```

- Allowed Values: "ON" | "OFF"
- Default Value: "ON"

TagListAntennaCombine Examples	
Command	>TagListAntennaCombine?
Response	TagListAntennaCombine = ON
Command	>TagListAntennaCombine = off
Response	TagListAntennaCombine = OFF

TagListMillis

9900 | 9680 | 9650

Standard TagList data includes timestamps with second-resolution, but the reader actually records timestamps internally with millisecond-resolution. You use the TagListMillis command to enable the display of the millisecond data in the TagList timestamps.

- Allowed Values: "ON" | "OFF"
- Default Value: "ON"
- When TagListMillis is set to "On", the TagList timestamps are of the form: YYYY/MM/DD HH:MM:SS.mmm
- When TagListMillis is set "Off", the TagList timestamps are of the form: YYYY/MM/DD HH:MM:SS

TagListMillis Examples	
Command Response	>TagListMillis? TagListMillis = On
Command Response	>TagList? Tag:DEAD BEEF CAFE 8042, Disc:2006/05/30 12:11:57.388, Last:2006/05/30 12:11:57.388, Count:1, Ant:0, Proto:1
Command Response	>TagListMillis = Off TagListMillis = Off
Command Response	>TagList? Tag:DEAD BEEF CAFE 8042, Disc:2006/05/30 12:12:02, Last:2006/05/30 12:12:02, Count:1, Ant:0, Proto:1

Clear TagList

9900 | 9680 | 9650

The "Clear TagList" command instructs the reader to immediately clear its internal TagList.

Clear TagList Examples	
Command Response	>Clear TagList TagList has been cleared!

TagStreamMode

9900 | 9680 | 9650

The TagStreamMode command turns on or off the TagStream functionality. Rather than having to continually poll the reader to fetch tag data, or rely on the NotifyMode functions, which entail some latency, you can now configure the reader to stream Tag read events to a TCP socket or the reader's serial port.

When TagStreamMode is on, each and every tag read causes the reader to stream the tag information to the TagStreamAddress (see below). There is no buffering of data, or combining of data from multiple antennas, as in the TagList. If a single inventory sees the same tag three times, it streams three tag read events.

The format of the streamed data is specified by the TagStreamFormat and TagStreamCustomFormat commands (below).

- Allowed Values: "ON" | "OFF"
- Default Value: "OFF"
- If the reader is unable to successfully stream the data (TCP socket error, for instance), ***the stream data is lost***.

TagStreamMode Examples	
Command Response	>TagStreamMode? TagStreamMode = OFF
Command Response	>TagStreamMode = on TagStreamMode = ON

TagStreamAddress

9900 | 9680 | 9650

The TagStreamAddress command specifies where TagStream event data should be sent. There are two delivery methods supported, as shown in the table below:

TagStreamAddress	Description
hostname:port	Send a message to a specified port on a networked machine. The address takes the form "hostname:port." For example, "123.01.02.98:3450" or "listener.alientechnology.com:10002"
SERIAL	Send a message to the serial port. The word "serial" is used as the address. The word is not case sensitive.

- Note: care must be taken when streaming data to the serial port. Tag read events can happen very fast, and may have a deleterious effect on the responsiveness of the serial port when issuing commands at the same time.

TagStreamAddress Examples	
Command	`>TagStreamAddress?`
Response	`TagStreamAddress = 10.1.0.12:4001`
Command	`>TagStreamAddress = serial`
Response	`TagStreamAddress = SERIAL`

TagStreamFormat

9900 | 9680 | 9650

The TagStreamFormat command lets you specify the formatting of TagStream data (separate from the TagList). The command takes a text string as its argument, and can be one of the following:

Format	Description
Text	TagStream data is displayed as plain text messages, one Tag event per line.
Terse	TagStream data is displayed as plain text messages, one Tag event per line, but with a simplified output showing just the TagID, Antenna, and ReadCount, with no labels.
XML	TagStream data is displayed in XML text format
Custom	TagStream data is displayed in the format given by TagStreamCustomFormat.

- The default value is "Terse".
- The formatting of TagStream data is the same as for TagList data. See the description of TagListFormat for output formats and examples of each type.

TagStreamFormat Examples	
Command	`>TagStreamFormat?`
Response	`IOStreamFormat = Terse`
Command	`>TagStreamFormat = Text`
Response	`TagStreamFormat = Text`

TagStreamCustomFormat

9900 | 9680 | 9650

The TagStreamCustomFormat command allows a customized TagStream format to be defined. Once the format has been defined it can be applied by issuing the command "`TagStreamFormat = Custom`".

The TagStreamCustomFormat command takes a single text line argument that defines how each Tag event should be represented. This argument can be made up of a mixture of text and custom tokens, as defined in the table below. The maximum length of a TagStreamCustomFormat definition is 255 characters.

When the reader outputs a TagStream event, the tokens and variables in the custom format are replaced with the actual values for the Tag event.

- The available custom tokens for the TagStream are the same as for TagList. See the TagListCustomFormat description for a table of allowed tokens.
- The default TagStreamCustomFormat is "%k "

TagStreamCustomFormat Examples	
Command	`>TagStreamCustomFormat?`
Response	`TagStreamCustomFormat = %k`
Command	`>TagStreamCustomFormat = Tag:%i Ant:%a`
Response	`TagStreamCustomFormat = Tag:%i Ant:%a`

TagStreamKeepAliveTime

9900 | 9680 | 9650

When the reader sends a TagStream event message out over the network, it needs to open a TCP socket. Opening and closing a socket entails some processor and network overhead, and if the messages are coming out of the reader rapidly, this overhead can become a burden that hinders reader responsiveness.

The TagStreamKeepAliveTime command sets the amount of time (in seconds) that the reader keeps its TagStream socket open. For rapid TagStream events, it may be advantageous for the reader to hold the socket open continuously. You do this by setting TagStreamKeepAliveTime to some value greater than the expected time between events.

On the other hand, holding the socket open places a burden on the network. For infrequent TagStream events, it is advantageous to leave the TagStreamKeepAliveTime small so that the socket is opened only long enough for a single message to be sent out, and is then closed automatically after message delivery.

- Allowed Values: Integer(0..32767), default is 30
- The reader automatically reopens a socket that has timeout the next time it needs to send data.

TagStreamKeepAliveTime Examples	
Command Response	>TagStreamKeepAliveTime? TagStreamKeepAliveTime = 30
Command Response	>TagStreamKeepAliveTime = 2 TagStreamKeepAliveTime = 2

StreamHeader

9900 | 9680 | 9650

When the reader initially opens a TCP socket for an IOStream or TagStream, it optionally outputs some header information, which serves to identify the reader to the receiving application, and specifies whether the stream is a TagStream or IOStream.

- Allowed Values:"ON" | "OFF". default is “ON”
- The StreamHeader command applies to both the IOStream and TagStream.
- The TagStream header is the same as the IOStream header except for the first line.
- The StreamHeader is only sent at the start of a socket connection. As long as the socket remains open, no further headers are sent. If the socket closes and is reopened, a new StreamHeader is sent.
- The StreamHeader is included when streaming to the serial port.

Following is an example header from the start of an IOStream. This format is used for the Text, Terse, and Custom IOStreamFormats. XML stream headers contain the same data, wrapped in XML tags.

```
#Alien RFID Reader I/O Stream\r\n
#ReaderName: Alien RFID Reader\r\n
#Hostname: alien-10116A\r\n
#IPAddress: 192.168.1.100\r\n
#CommandPort: 23\r\n
#MACAddress: 00:80:66:10:11:6A\r\n
#Time: 2006/12/18 11:19:26\r\n\0
```

StreamHeader Examples	
Command Response	>StreamHeader? StreamHeader = ON
Command Response	>StreamHeader = off StreamHeader = OFF

Macro Commands

The reader supports on-board stored "macros", which are named sequences of commands that may be run as a group with a single reader command. Macros are text files with a .arm (Alien Reader Macro) extension that can contain reader commands, empty lines, and comment lines (beginning with "#" or "//").

- Macro names are inferred from the macro's filename (minus the .arm extension), so they must adhere to certain restrictions – only alphanumeric characters (a-z, A-Z, 0-9) are allowed.
- Macro names are case-sensitive, so "myMacro" is distinct from "mymacro".
- After the reader starts up, it automatically runs any stored macro named "default". The reader ignores a "reboot" command within the "default" macro.

For example, a definition of a macro named "myMacro":

```
# myMacro.arm
# I made it sometime in January

# Start out at the beginning
FactorySettings

# Set the readername
ReaderName = MyTestReader

# Set up AutoMode
AutoStartTrigger = 1 2
AutoMode = on
```

Macros may be created externally, then uploaded to the reader, through the web interface, or they can be created on-the-fly from the reader's command line interface with a handy macro recording feature.

Once macros are loaded onto the reader, they persist through rebooting cycles and even reader upgrades. Macros can be listed, viewed, run, or removed from the reader with the new set of Macro commands. When a macro is run, the reader executes each line in the macro file as if you had entered it explicitly on the command line.

MacroList

9900 | 9680 | 9650

You use the MacroList command so see a list of all the macros currently residing in the reader, one macro name per line.

- If there are no macros installed, the reader responds with, "(No Macros Installed)".
- Two example macros, named "example1" and "example2" may already be found in your reader, and may be updated in future releases. If you create your own macros with these same names, they may be replaced during an upgrade.

MacroList Examples

Command Response	>MacroList example1 example2
Command Response	>MacroList (No Macros Installed)

MacroView

9900 | 9680 | 9650

You use the MacroView command, followed be the name of an installed macro, to view the contents of a macro.

- If the specified macro name does not correspond to an installed macro, then the reader responds with, "Error 28: Unknown error. There is no readable macro with that name."

MacroView Examples	

Command
Response
	>MacroView example1
	// Example 1 Macro
	
	// Look for all types, small number of tags expected
	RFModulation = STD
	TagType = 31
	AcquireMode = Global Scroll
	
	// We'll want to see a notification when new tags are seen
	NotifyAddress = 10.10.82.50:3600
	NotifyTrigger = Add
	
	// Set up AutoMode to run very quickly
	AutoModeReset
	PersistTime = 10
	AutoStopTimer = 100
	
	// Go!
	NotifyMode = On
	AutoMode = On

Command Response	>MacroView foo Error 28: Unknown error. There is no readable macro with that name.

MacroDel

9900 | 9680 | 9650

MacroDel, followed by the name of an installed macro, permanently deletes the named macro.

- No warning or second chance is given.
- If the specified macro name does not correspond to an installed macro, then the reader responds with, "Error 28: Unknown error. Unable to delete - there is no macro with that name."

MacroDel Examples	
Command Response	>MacroDel example1 Macro deleted.
Command Response	>MacroDel foo Error 28: Unknown error. Unable to delete - there is no macro with that name.

MacroDelAll

9900 | 9680 | 9650

MacroDelAll permanently deletes all of the installed macros.

- No warning or second chance is given.

MacroDelAll Example	
Command	>MacroDelAll
Response	All Macros Deleted

MacroRun

9900 | 9680 | 9650

MacroRun, followed by the name of an installed macro, executes the commands given in the named macro. When a macro is run, the reader scans the associated .arm file and executes each of the contained within (skipping empty lines, and lines beginning with "#" or "//").

You must take care to ensure that the list of commands in the macro is sufficient to bring a reader from any arbitrary configuration to the desired state. For instance, it may not be sufficient to just include the command, "AutoMode = On", without first explicitly setting the other AutoMode properties.

- As the reader executes each line of the macro, it displays the command being issued, but not the response to each command.
- Upon successful execution of a macro, the reader responds with, "Macro Run Successfully".
- If the specified macro name does not correspond to an installed macro, then the reader responds with, "Error 28: Unknown error. There is no readable macro with that name."
- If any command within a macro generates an error condition, the reader responds with a descriptive message and halts further execution of the macro. The offending line is included in the error response.
- After the reader starts up, it automatically runs any stored macro named "default". The reader ignores a "reboot" command within the "default" macro.

MacroRun Examples	
Command	>MacroRun example1
Response	RFModulation = STD TagType = 31 AcquireMode = Global Scroll NotifyAddress = 10.10.82.50:3600 NotifyTrigger = Add AutoModeReset PersistTime = 10 AutoStopTimer = 100 NotifyMode = On AutoMode = On Macro Run Successfully
Command	>MacroRun macro1
Response	ExternalOutput = foo Error 28: Unknown error. Macro error handling line: ExternalOutput = foo
Command	>MacroRun foo
Response	Error 28: Unknown error. There is no readable macro with that name.

MacroStartRec
MacroStopRec

9900 | 9680 | 9650

MacroStartRec, followed by a macro name, opens a new macro file based on the supplied name and writes to this macro file all of the subsequent commands that you enter. Commands are recorded until you issue the MacroStopRec command or another MacroStartRec command. The newly-created macro is then available for use – just as if you had created it externally and uploaded it.

The response to the MacroStartRec command is an acknowledgment with brief instructions on how to proceed.

- Recording a macro replaces any other macro with the same name.
- While recording, the "Alien>" prompt changes to "Alien>R>" to indicate you are in recording mode.
- Macro names are restricted to only the alphanumeric characters – a-z, A-Z, 0-9.
- It is not possible to run a macro within another macro.

MacroStartRec & MacroStopRec Examples	
Command Response Command Response	>MacroStartRec recordedMacro1 MacroStartRec = Recording a new macro. Enter commands, then use MacroStopRec to stop. Alien>R>ReaderName = Recorded Macro 1 ReaderName = Recorded Macro 1
Command Response Command Response Command Response	Alien>R>MacroStopRec MacroStopRec = Success! Alien>MacroList example1 example2 recordedMacro1 Alien>MacroView recordedMacro1 ReaderName = Recorded Macro 1

Acquire Commands

Acquire commands allow you to configure the reader's parameters that govern how it utilizes the various air protocols to best read a particular tag population.

AcquireMode

9900 | 9680 | 9650

When the reader is called upon to read a tag it does so using the current AcquireMode. Currently the allowable modes are as follows:

AcquireMode	Description
Inventory	Perform full inventory of multiple tags (recommended)
Global Scroll	Perform fast search for single tag (not recommended)

The default setting is Inventory. For a detailed description of the different modes, please refer to the earlier chapter titled "Tag Fundamentals". Global Scroll is generally not recommended.

INVENTORY

The Inventory acquire mode performs a full anti-collision search on tags in the reader's field of view. This method locates and distinguishes multiple tags in front of the reader at the same time.

GLOBAL SCROLL

The Global Scroll acquire mode instructs the reader to read a single tag repeatedly. This is a very fast tag reading method that is most effective when only one tag at a time is expected to be within reader range, as in conveyor belt applications. Under such circumstances, the performance for single tag reading is considerably faster than repeatedly doing a full tag search using the Inventory mode.

NOTE: If multiple tags are in range of the reader when this mode is used, the reader either selects one of the tags (usually the "strongest'" or "loudest") to read and report, or reads none of the tags.

AcquireMode Examples	
Command	`>AcquireMode = Global Scroll`
Response	`AcquireMode = Global Scroll`
Command	`>AcquireMode = Inventory`
Response	`AcquireMode = Inventory`

TagType

9900 | 9680 | 9650

There are currently five types of tags supported by Alien RFID readers, each implementing one of the three EPCglobal air protocols.

By allowing the reader to ignore certain tag types, the reader can focus its attention and spend more time acquire tags at a higher speed. This is accomplished by setting the TagType property. The TagType value is a bitmap, where each bit enables a certain tag type, as summarized in the following table.

TagType Bit	Tag Enabled	Air Protocol	9900	9680	9650
Bit 1	"Quark"	Class 1 / Gen 1	-	-	-
Bit 2	"Omega"		-	-	-
Bit 3	"Lepton"		-	-	-
Bit 4	C0	Class 0	-	-	-
Bit 5	C1G2	Class 1 / Gen 2	✓	✓	✓
Bit 6	reserved		-	-	-
Bit 7	reserved		-	-	-
Bit 8	reserved		-	-	-

- All current shipping readers only support Class 1 / Gen 2, so the only allowed TagType is 16.

TagType Examples	
Command	>TagType = 16
Response	TagType = 16
Command	>TagType?
Response	TagType = 16

AcqG2Cycles

9900 | 9680 | 9650

AcqG2Cycles takes a single integer parameter between 1 and 255. It is the number of acquisition cycles that are performed each time the reader scans for Class1 / Gen2 tags.

Note: While this attribute has a maximum value of 255 (as do the other acquisition settings), setting them to high values can result in very long acquisition times, which may cause the reader to appear non-responsive. For example, setting both AcqG2Cycles and AcqG2Count to 255 causes the reader to perform more than 65,000 acquisitions when it is directed to look for Class1 / Gen2 tags.

The AcqG2Cycles parameter controls the "outer loop" of the Class1 / Gen2 acquisition cycle, described in the "Tag Reading Fundamentals" chapter. Its value has a significant impact on the time the reader takes to perform each scan for tags.

- Allowed Values: Integer(0..255)
- Default Value: 1
- Read chapter 3 for more information on how this parameter affects tag inventories.

AcqG2Cycles Examples	
Command	>AcqG2Cycles?
Response	AcqG2Cycles = 1
Command	>AcqG2Cycles = 2
Response	AcqG2Cycles = 2

AcqG2Count

9900 | 9680 | 9650

AcqG2Count takes a single integer parameter between 1 and 255. It is the number of reads (Global Scroll or Inventory) that are performed in each Class1 / Gen2 acquisition cycle.

For example, if AcqG2Count is set to 10, then ten acquisition commands are issued during each acquisition cycle.

- Allowed Values: Integer(0..255)
- Default Value: 3
- Read chapter 3 for more information on how this parameter affects tag inventories.

AcqG2Count Examples	
Command	>AcqG2Count = 10
Response	AcqG2Count = 10
Command	>AcqG2Count?
Response	AcqG2Count = 10

AcqG2Q

9900 | 9680 | 9650

AcqG2Q takes a single integer parameter between 0 and 7. It is the starting "Q" value used to tune the performance of the Class1/Gen2 air protocol. For example, if AcqG2Q is set to 3, then the reader starts looking for tags with Q = 3. The reader may tune the active Q value up or down during an inventory (*not* when using AcquireMode=Global Scroll), but always starts with this value.

Small Gen2 tag populations benefit from a small Q value (0-1), while larger Gen2 tag populations benefit from a higher Q value (2-5).

- Allowed Values: Integer(0..7)
- Default Value: 3
- Read chapter 3 for more information on how this parameter affects tag inventories.

AcqG2Q Examples	
Command	>AcqG2Q = 1
Response	AcqG2Q = 1
Command	>AcqG2Q?
Response	AcqG2Q = 1

AcqG2QMax

9900 | 9680 | 9650

During Gen2 inventories, the reader starts off with a Q value equal to AcqG2Q, but then increases or decreases the Q value according to how many tags it is finding in the field. The AcqG2QMax allows you to place an upper limit on the sliding Q value.

- Allowed Values: Integer(0..15)
- Default Value: 7

AcqG2QMax Examples	
Command Response	>AcqG2QMax? AcqG2QMax = 7
Command Response	>AcqG2QMax = 3 AcqG2QMax = 3

AcqG2Select

9900 | 9680 | 9650

AcqG2Select takes a single integer parameter between 0 and 255. It is the number of SELECT commands issued at the start of each Class1/Gen2 inventory cycle, used to tune the performance of the Class1/Gen2 air protocol.

At the start of each G2 inventory cycle, the reader issues a SELECT command, which serves to place tags matching the current mask into the "un-inventoried" state, similar to a WAKE command in Class1/Gen1. The AcqG2Select command allows you to specify whether or not to issue SELECT commands, and how many. If no Selects are issued, then your AcqG2Mask will have no effect.

When attempting to inventory extremely large, static tag populations, it may be helpful to disable SELECTs, so that easy-to-read tags remain in the inventoried state, allowing the reader to focus on the hard-to-read tags.

- Allowed Values: Integer(0..255)
- Default Value: 1
- Setting AcqG2Select = 0 causes the reader to not issue any SELECTs.
- Read chapter 3 for more information on how this parameter affects tag inventories.

AcqG2Select Examples	
Command Response	>AcqG2Select? AcqG2Select = 1
Command Response	>AcqG2Select = 0 AcqG2Select = 0

AcqG2Session

9900 | 9680 | 9650

AcqG2Session takes a single integer parameter between 0 and 3. It is the inventory session used by the reader when acquiring Class1/Gen2 tags.

The Class1/Gen2 protocol allows tags to maintain an "inventoried state" for each of four sessions, numbered 0 through 3. Theoretically, multiple readers, each operating on different sessions, can simultaneously interrogate the same population of tags (in practice, though, tags tend to operate best when only one reader at a time is transmitting). The AcqG2Session command specifies which session the reader operates on. Another reason to use different sessions is that tags maintain the "inventoried" state for various "persistence" times, which can be helpful for different use cases.

G2 Session	Nominal Persistence
0	Tag energized: Indefinite Tag not energized: None
1	Tag energized: 500 ms – 5 sec Tag not energized: 500 ms – 5 sec
2	Tag energized: Indefinite Tag not energized: > 2 sec
3	Tag energized: Indefinite Tag not energized: > 2 sec

- The default value for AcqG2Session is 1.
- Read chapter 3 for more information on how this parameter affects tag inventories.

AcqG2Session Examples	
Command	>AcqG2Session?
Response	AcqG2Session = 1
Command	>AcqG2Session = 2
Response	AcqG2Session = 2

G2Wake

9900 | 9680 | 9650

The G2Wake command issues a Gen2 "Select" command to the field of tags. You can optionally specify the number of selects to issue. This command can be useful when tags are inventoried in long sessions (2 or 3) with no selects during the inventory, leaving the tags unresponsive to inventories to long periods of time. Issuing a G2Wake command directs those tags to participate again in the next inventory round.

If you follow the G2Wake command with an integer between 1 and 255, the reader issues that many Select commands.

G2Wake Examples	
Command	>G2Wake
Response	OK
Command	>G2Wake 10
Response	OK

AcqG2Mask

9900 | 9680 | 9650

The AcqG2Mask command allows you to specify the mask used when inventorying Class1/Gen2 tags. Masks are important in both addressing tags and interrogating them. For a detailed description of masks, please refer to the earlier chapter entitled "Tag Reading Fundamentals".

The AcqG2Mask command in its basic form takes four parameters:

- memory bank to mask on (1-3)
- bitPtr into the memory bank, as a decimal number (0-2097151)
- bitLen of mask, as a decimal number (0-255)
- array of Hex Bytes separated by white spaces

The AcqG2Mask command requires the bank field, which is restricted to banks 1-3 (masking in the bank 0 is not permitted by the G2 protocol).

G2 bank 0 - RESERVED (not maskable)
G2 bank 1 - EPC (CRC + PC + EPC)
G2 bank 2 - TID (Tag identifier)
G2 bank 3 - USER (user-specified data, not supported by all tags)

You may optionally specify up to four separate Class1/Gen2 masks using this command. Only those tags matching all of the requested masks participate in an inventory. If the AcqG2MaskAction is "exclude", then only those tags matching none of the requested masks participates in an inventory. Each mask entry is separated from the previous entry with a vertical bar (|).

Compatibility Note:
The older AcqMask (or just "Mask") commands are still present for backward compatibility reasons. Using the older AcqMask or Mask commands (which only act on the EPC bank, and always start at the beginning of the EPC data) will still work, and the AcqG2Mask parameter will then report the equivalent Gen2-style mask. Similarly, setting the AcqG2Mask will also set the AcqMask and Mask parameters to the corresponding mask value, but only if the AcqG2Mask refers to a valid portion of EPC data (not the CRC or PC words). It is therefore possible to have a valid AcqG2Mask, but a Mask/AcqMask of zero (for example, if the AcqG2Mask is on a non-EPC bank). The reader will always use the AcqG2Mask settings.

- Setting the AcqG2Mask = 0 (or “All”) addresses all Gen2 tags currently in the RF field.
- The data bytes given must supply at least the number of bits given by bitLen.
- Mask data is read from the given bytes from MSB to LSB, so if the bitLen doesn't fall on an 8-bit boundary, the remaining bits (in the last byte of data) must be the most-significant bits.
- The special case where bitLen=0 (you give a mask that has zero length) causes tags to respond only if the memory location pointed to by bitPtr is a valid memory location.

AcqG2Mask Examples	
Command Response	// Clearing the AcqG2Mask >AcqG2Mask = 0 AcqG2Mask = 00
Command Response	// Mask for any tag with EPC starting with "03" >AcqG2Mask = 1, 32, 8, 03 AcqG2Mask = 1 32 8 03
Command Response	// Mask for only Alien tags (TID bank, byte #3=0x34 >AcqG2Mask = 2, 16, 8, 34 AcqG2Mask = 2 16 8 34
Command Response	// Mask only tags with User memory (zero-length mask, no data!) >AcqG2Mask = 3, 0, 0 AcqG2Mask = 3 0 0

Command
Response
	// Mask only tags starting with "DE", with User
	// memory (multiple masks)
	>AcqG2Mask = 1, 32, 8, DE | 3, 0, 0
	AcqG2Mask = 1 32 8 DE|3 0 0

AcqG2MaskAction

9900 | 9680 | 9650

The Class1/Gen2 protocol allows masks to be inclusive (only tags matching the mask respond) or exclusive (only tags not matching the mask respond). If you are using multiple masks, the reader will automatically use the appropriate Gen2 actions under the hood to get the masks do what you want, or you can use an extended version of the AcqG2MaskAction command and give all of the Gen2 protocol details yourself.

If you choose to specify the mask actions yourself, then the actions are listed (separated by vertical bars, '|') in such a way that the first action in the list corresponds to the first mask in the list of masks, and so on. When using this "state aware" masking, the actions that you specify can either be numeric (0-7) or textual (A-, AB, -B, etc.). Extra action entries without corresponding masks will be ignored, and extra mask entries without corresponding actions will continue to use the last action in the list.

The mask action indicates what you want to do with the "inventoried" flag in the tag (move it to the A state, move it to the B state, leave it alone, or swap its state), for those tags that match the mask, as well as those tags that don't match the mask. The mask can change the "inventoried" flag for a completely different Session than is currently being used, and you can even restrict each mask to only operate on specific antennas.

The allowed values for AcqG2MaskAction are:

Up to 4 |-separated values consisting of:

`0 | 1 | 2 | 3 | 4 | 5 | 6 | 7` - Actions defined by C1G2 – see below

`AB | A- | -B | s- | BA | B- | -A | -s` - Textual equivalents to numeric values.

Or one of:

"Include" – same as "0 |1 |1 |1" or "AB | A- | A- | A-"

"Exclude" – same as "4 | 5 | 5 | 5" or "BA | B- | B- | B-"

Remember, "move to A" generally means the tag will be ready to participate in the inventory, and "move to B" means the tag will not participate in the inventory. If you've changed the AcqG2Target to something other than "A", then your inventories will behave differently.

C1G2 Action	Text Version	Description of Action	
		Matching Tags	Non-Matching Tags
0	**AB**	assert SL or Inventoried→A	deassert SL or Inventoried→B
1	**A-**	assert SL or Inventoried→A	do nothing
2	**-B**	do nothing	deassert SL or Inventoried→B
3	**s-**	negate SL or Inventoried A←→B	do nothing
4	**BA**	deassert SL or Inventoried→B	assert SL or Inventoried→A
5	**B-**	deassert SL or Inventoried→B	do nothing
6	**-A**	do nothing	assert SL or Inventoried→A
7	**-s**	do nothing	negate SL or Inventoried A ←→ B

You can use any combination of numeric or textual versions of the actions. The special values "Include" and "Exclude" correspond to the following numeric and textual values:

Special Action	Numeric Version	Textual Version	Description
Include	**0|1|1|1**	**AB|A-|A-|A-**	On first mask, select all matching and deselect all non-matching. For all subsequent masks, select matching and leave non-matching alone
Exclude	**4|5|5|5**	**BA|B-|B-|B-**	On first mask, deselect all matching and select all non-matching tags. For all subsequent masks, deselect matching and leave non-matching alone.

Since the last action will be used for any additional masks, the Include type could have been specified with just "0|1" and the Exclude type could have been specified with just "4|5". The reader will help you out

even more: if you give just a single mask action of 0, it will automatically make any other masks have an action of 1 (not 0) – making it act like the true "Include" case. Similarly, if you give just a single mask action of 4, the reader will automatically make any other masks have an action of 5 (not 4) – making it act like the true "Exclude" case.

AcqG2MaskAction Examples	
Command Response	>AcqG2MaskAction? AcqG2MaskAction = Include
Command Response	>AcqG2MaskAction = Exclude AcqG2MaskAction = Exclude
Command Response	>AcqG2MaskAction = AB | A- AcqG2MaskAction = AB | A-
Command Response	>AcqG2MaskAction = 0 | 1 AcqG2MaskAction = 0 | 1

AcqG2MaskAntenna

9900 | 9680 | 9650

The AcqG2MaskAntenna command accepts up to four values, with a one-to-one correspondence between entries in the AcqG2MaskAntenna list and entries in the AcqG2Mask list. Extra antenna entries without corresponding masks will be ignored, and extra mask entries without corresponding antennas will continue to use the last antenna entry in the list.

Each of the values in the AcqG2MaskAntenna is a bitmap of which antennas the corresponding mask will act on. The four antennas are each selected with the first 4 low-order bits (bit #1 corresponds to antenna 0, bit #2 corresponds to antenna 1, etc.). Of course, you could theoretically pick (and OR together) any combination of these. For instance, to cause the mask to act on all four antennas, the antenna value would be 0x0F (00001111).

Antenna Value	Binary Equiv.	Selected Antenna
0x01	00000001	Antenna 0
0x02	00000010	Antenna 1
0x04	00000100	Antenna 2
0x08	00001000	Antenna 3

- Values supplied must be a 01-0F.
- The default AcqG2MaskAntenna is 0F (all antennas).

AcqG2MaskAntenna Examples	
Command Response	>AcqG2MaskAntenna? AcqG2MaskAntenna = 0F
Command Response	>AcqG2AccessPwd? AcqG2AccessPwd = 01 02 03 04
Command Response	// 1st mask on Ant #0, 2nd mask on Ant #1, 3rd mask on Ant #2, etc. >acqg2maskantenna = 1 | 2 | 4 | 8 AcqG2MaskAntenna = 01 | 02 | 04 | 08

AcqG2SL

9900 | 9680 | 9650

The SL flag operates in a fashion similar to a session's "inventoried" flag, but there is only one SL flag, and it is accessible from any session. Mask actions can modify the state of the SL flag, and the SL flag

can also be changed during an inventory. SL flags have persistence, just like session inventoried flags do, and most Alien tags have somewhat long SL persistence times -- on the order of minutes.

The default value of AcqG2SL is “all”, which tells the reader to ignore the SL flag when inventorying tags. If AcqG2SL = SL, then only tags with an asserted SL flag are inventoried, and if AcqG2SL = nSL, then only tags with a deasserted SL flag are inventoried.

- Value supplied must be “SL”, “nSL”, or “ALL”
- The default AcqG2SL is ALL.

AcqG2SL	
Command Response	>AcqG2SL? AcqG2SL = All
Command Response	// Only tags with SL asserted will be inventoried >AcqG2SL = SL AcqG2SL = SL

AcqG2AccessPwd

9900 | 9680 | 9650

The AcqG2AccessPwd value is used when the programming operation occurs on a password-protected Class 1/Gen2 tag. If such a tag has a non-zero Access Pwd written to it, the reader must supply the same password in order to place the tag into the "secured" state – a state which allows locking, unlocking, and write operations. If the provided AcqG2AccessPwd doesn't match the tag's stored Access Pwd, then programming operations on that tag may fail with the "Access failed" error message.

This command is different from the similar-sounding ProgG2AccessPwd command. ProgG2AccessPwd is a placeholder for an Access Pwd value to be written to the tag, whereas the AcqG2AccessPwd provides the tag's current Access Pwd to be used when operating on a Class 1/Gen2 tag that is already password-protected.

- Value supplied must be a 4-byte value.
- The default AcqG2AccessPwd is 00 00 00 00.

AcqG2AccessPwd Examples	
Command Response	>AcqG2AccessPwd = 1 2 3 4 AcqG2AccessPwd = 01 02 03 04
Command Response	>AcqG2AccessPwd? AcqG2AccessPwd = 01 02 03 04

AcqG2Target

9900 | 9680 | 9650

When the reader inventories a Gen2 tag, it flips an A/B state flag in the tag that indicates whether that tag has been inventoried yet. The default behavior is for the reader to "select" all tags over to the A state, and move them to the B state as they are being inventoried.

You can change this behavior to alternatively start the tags out in the B state and move them to the A state, or just flip-flop between the A and B states from one inventory to the next (generally producing more redundant tag reads).

The effect that changing the AcqG2Target mode has on tag inventories depends heavily on the particular AcqG2Session in use, read zone control, time the tags are in the field, etc. You should have a firm grasp of the Gen2 protocol before you change this setting.

- Allowed Values: "A" | "B" | "AB".

- Default Value: "A".

AcqG2Target Examples	
Command Response	>AcqG2Target? AcqG2Target = A
Command Response	>AcqG2Target = AB AcqG2Target = AB

AcqG2TagData

9900 | 9680 | 9650

The AcqG2TagData command allows you to specify additional portions of Gen2 tag memory to fetch during an inventory (passwords, TID, User memory, etc.). The reader fetches the additional data for each tag it reads, and includes that data in the TagList. This can significantly impact read performance, but it is still much faster than doing an inventory and then individually querying each tag for the desired data.

Additional tag data blocks to be fetched are specified by bank, word pointer, and word length values. Up to four tag memory data blocks can be specified, separated by vertical bars (|). If you specify a word length of zero, the reader performs a "zero-length read", which returns all of the memory after the word pointer (see "G2Read", later in this document).

If the requested tag memory data block does not exist or if the reader fails to read it (for example, due to the noise) the data block is not reported in the TagList.

Setting `AcqG2AccessPwd` allows reading data from password protected memory banks. For example, you can password protect sensitive data in User memory bank using the `HideAlienUserBlocks` command and then read the data during inventories by setting the correct AcqG2AccessPwd password.

If any AcqG2TagData has been acquired, then that data is included automatically in the text- and XML-formatted TagLists, prefixed with "D1", "D2", etc. labels. Tag memory data blocks can also be included in custom-formatted TagLists by using the ${G2DATA1}, ${G2DATA2}, ${G2DATA3}, and ${G2DATA4} tokens (refer to the TagListCustomFormat description).

- Setting up AcqG2TagData causes the reader to perform extra work while reading tags, which may impact read performance.
- The format of each data block is the same as you use in the G2Read command, so you can determine what G2Read command gives you the desired data, and use that to configure AcqG2TagData.
- To disable AcqG2TagData, simply set it to 0. This is the default value.

AcqG2TagData Examples	
Command Response	>AcqG2TagData? AcqG2TagData = 0
Command Response Command Response	// Show the PC word and the 1st word of the EPC (bank 1, words 1-2) >AcqG2TagData = 1 1 2 AcqG2TagData = 1 1 2 >t Tag:0000 0000 0000 0000 0000 0001, …, D1:3000 0000 Tag:DEAD BEEF CAFE 8042 55AA 0006, …, D1:3000 DEAD Tag:AABB 3344 DDEE 7788 9900 1132, …, D1:3101 AABB
Command Response Command Response	// Show all of User memory too >AcqG2TagData = 1 1 2 | 3 0 0 AcqG2TagData = 1 1 2 | 3 0 0 >t Tag:0000 0000 0000 0000 0000 0001, …, D1:3000 0000, D2:1FF1 1FF2 Tag:DEAD BEEF CAFE 8042 55AA 0006, …, D1:3000 DEAD Tag:AABB 3344 DDEE 7788 9900 1132, …, D1:3101 AABB // (some tags don't have User memory)

AcqG2AntennaCombine

9900 | 9680 | 9650

The AcqG2AntennaCombine command controls how the Gen2 SELECT command is issued during the inventory cycle.

When set to ON (the default value), the reader first issues the Gen2 SELECT command on all antennas in the AntennaSequence, and then the proceeds with the inventory loops. This setting might be effective in the situation when there are many tags in the field, so you would "wake them up" first and then inventory them (with a long session - 2 or 3), so there would be no redundant reads from different antennas.

When set to OFF, the Gen2 SELECT command is issued on each antenna as they are used. This setting will increase number of reads which might help, for example, the conveyor/speed use cases. It also gives better support for separate read zone use cases, where you need to that a tag passed from one antenna's zone to the next. Each antenna gets its own chance to read the tag population. This is also essential if you are using the Speed and Direction data on different antennas to determine a tag's location or direction of motion.

- Allowed Values: "ON" | "OFF"
- Default Value: "ON"

AcqG2AntennaCombine Examples	
Command Response	>AcqG2AntennaCombine? AcqG2AntennaCombine = ON

	// set reader parameters
	>AntennaSequence = 0 1
	AntennaSequence = 0 1
	>AcqG2Session = 1
	AcqG2Session = 1
	>TagListAntennaCombine = OFF
	TagListAntennaCombine = OFF
	
	// first, issue selects on all antennas,
	// then perform an inventory on all antennas

Command Response	>AcqG2AntennaCombine = ON AcqG2AntennaCombine = ON >t Tag:8000 8004 2665 8426, Count:1, Ant:0 // the tag was only inventoried on the first antenna
	// perform 'select-inventory' sequence on each antenna
Command Response	>AcqG2AntennaCombine = OFF AcqG2AntennaCombine = OFF >t Tag:8000 8004 2665 8426, Count:1, Ant:0 Tag:8000 8004 2665 8426, Count:1, Ant:1 // tag was selected and inventoried on both antennas

AcqG2Ops

9900 | 9680 | 9650

The primary task of performing an inventory on a population of tags is to determine each tag's unique EPC code. You could also use the AcqG2Data command to instruct the reader to perform additional operations during inventories, for example, read other portions of tag memory (Access & Kill passwords, TID and USER memory banks), and report them back to you as additional data fields in the taglist.

With the new AcqG2Ops command, you can perform write, lock, and even kill operations, right in the middle of an inventory, on multiple antennas. Additional operations support the new BlastWrite feature of Alien Higgs4 tags, which allow you to perform those operations on an entire tag population at the same time.

You can define up to eight separate operations, and while the reader has each tag singulated during an inventory, it will perform the list of operations on that tag. Once the operations are done (or one of them fails), the reader will begin communication with the next tag in the inventory, and perform the same operations on that tag.

AcqG2Ops are numbered, from 1 to 8, and are executed in numerical order. You can skip sequence numbers, and delete or define individual operations at any time. Once you have defined your AcqG2Ops, you enable them in one of two ways:

1. Set AcqG2OpsMode = On
    a. Default is "Off".
    b. When turned on, all tag inventory operations done by the reader will execute the AcqG2Ops: "t", "acquire", AutoMode running, etc.
2. Issue the new "to" (taglist ops) command, instead of "t".
    a. This executes the AcqG2Ops, even if AcqG2OpsMode=Off.

b. You can switch back and forth between “to” and “t”, to perform inventories with and without the AcqG2Ops being executed.

ACQG2OPS COMMAND SYNTAX

The general syntax to set a particular AcqG2Op is:

```
AcqG2Ops = <opsNum> <opsAction> <opsArgs>
```

<opsNum> is the number for this op (1-8).
<opsAction> indicates the name (or associated 1-byte code) of the operation action.
<opsArgs> is the space-separated list of arguments required by the given <opsAction>.

Following is a list of operations currently supported, along with their arguments. Numeric arguments that are supplied as hex values have the number of bytes before the argument name (e.g. <2:blockMask1> is a two-byte hex value), otherwise they are decimal values. For details on the operation actions, arguments, and data/results returned by each of these G2Ops, see the descriptions, following.

Gen2 Action Names	Code	Arguments
G2Read	01	<bank> <wordPtr> <wordCount>
G2Write	02	<bank> <wordPtr> <2n:data…> [+, -, ++]
G2Kill	03	<4:killPwd>
(reserved)	04	-
G2Lock	05	<field> <lockType>
G2BlockErase	06	<bank> <wordPtr> <wordCount>
(reserved)	07	-
G2BlockPermalock	08	<blockPtr> <2:blockMask1> [<2:blockMask2>…]
G2GetPermalock	09	<blockPtr>
Alien Gen2 Action Names	**Code**	**Arguments**
AlienG2UserReadlock	0A	<1:blockMask>
AlienG2BlastLock	85	<field> <lockType>
AlienG2BlastErase	86	<bank> <wordPtr> <wordCount>
AlienG2BlastWrite	87	<bank> <wordPtr> <2n:data…>
AlienG2BlastBlockPermalock	88	<blockPtr> <2:blockMask1> [<2:blockMask2>…]
(reserved)	89	-
AlienG2BlastUserReadlock	8A	<1:blockMask>
(reserved)	8B	-
AlienG2TagStatus	8C	<1:statusMask>

Where:

<bank> : 0 – 3
<wordPtr> : 0 – 2097151 (base 10)
<wordCount> : 0 – 32 (base 10)
[+, -, ++] : One of these will case the G2write to increment its data.
+ = after a successful write
- = after a failure to write
++ = always

<field> : 0 (killPwd), 10 (accessPwd), 1, 2, 3 (EPC, TID, USER banks)
<lockType> : Desired lock operation:
L or Lock
U or Unlock
PL or PermaLock
PU or PermaUnlock
<blockPtr> : 0 – 4064 Used to point to a block of data in User memory
<blockMask> : Used to indicate which User blocks are/should be enabled
1st significant bit = Block #1
2nd significant bit = Block #2, etc.
G2BlockPermaLock uses a 16-bit <2:blockMask>
AlienG2UserReadLock uses an 8-bit <1:blockMask>
<statusMask>: Bits indicate which status values to get from the tag
1 = request bank writeLocks (bit #1)
2 = request User block readLocks (bit #2)
3 = request writeLocks and readLocks (bits #1 and #2)

Because you can do so much more during an inventory, you may encounter some new bottlenecks/restrictions than previously seen. For instance, in the USA, the reader is required to hop to a new frequency every 400ms. This interrupts all communications with the tags and therefore forces any of the BlastWrite operations to finish in time for the next hop. Also, tags that are inventoried have an internal flag flipped from the A state to the B state, and since the extended inventory takes longer to perform, some of the earlier tags placed in the B state may "fall back into" the A state. This is a natural condition of the protocol, and depends on the AcqG2Session being used. We recommend using AcqG2Session = 2 (with a very long B persistence time) when performing tag operations with AcqG2Ops.

- When executing AcqG2Ops in an inventory, they are executed in the order of their <opsNum>.
- …except for the AlienG2BlastXXXX operations, which are always executed last. Since the "blast" operations require the reader to already know the list of EPCs, it is significantly more efficient to do those operations at the end of the inventory.
- If an operation fails, subsequent AcqG2Ops for that tag will not execute. The reader will continue the inventory with the next tag and the first configured AcqG2Ops.
- Results of these operations are obtained in the taglist be using the new ${G2Ops} and ${G2Ops1..8} custom taglist tokens.

AcqG2Ops Examples	
	// Set AcqG2Ops #1 (write "DEAD BEEF" to the start of USER memory)
Command	> AcqG2Ops = 1 g2write 3 0 DE AD BE EF
Response	AcqG2Ops = 1 g2write 3 0 DE AD BE EF
	// Set AcqG2Ops #3 (lock USER bank)
Command	> AcqG2Ops = 3 g2lock 3 lock
Response	AcqG2Ops = 3 g2lock 3 lock
	// Get all AcqG2Ops:
Command	> AcqG2Ops?
Response	1 g2write 3 0 DE AD BE EF
	3 g2lock 3 lock
	// See it all in the Info dump:
Command	> i acquire
Response	**********
	ACQUIRE COMMANDS
	**********
	…snip…
	AcqG2Ops = 1 g2write 3 0 DE AD BE EF
	AcqG2Ops = 3 g2lock 3 lock
	…snip…
	// Get just AcqG2Ops #1:
Command	> AcqG2Ops 1?
Response	1 g2write 3 0 DE AD BE EF
	// Clear AcqG2Ops #3:
Command	> AcqG2Ops = 3
Response	AcqG2Ops = 3
	// Clear all AcqG2Ops:
Command	> AcqG2Ops = 0
Response	AcqG2Ops = 0

ACQG2OPS RESULTS SYNTAX

Results from AcqG2Ops operations are maintained for each tag in the reader's TagList, and will be included with any TagLists delivered to you that use custom formatting, including one of the new ${G2Ops} or ${G2Ops1}…${G2Ops8} custom taglist tokens.

Each result block from an AcqG2Ops has the general syntax:

<1:opsActionCode> <1:opsResultCode> [<data>]

In other words, there's a single byte with the `<opsActionCode>` for the operation that was run, followed by a single byte with the result code, followed by any data that may have been requested from the operation (g2read, g2getpermalock, alieng2tagstatus, etc.). The `<opsActionCode>` is the same code give in the list of AcqG2Ops actions, above.

The `<opsResultCode>` for a successful operation is `00` - anything else is an error code corresponding to one of the (hex) DSP error codes listed in appendix F of this Reader Interface Guide.

The `<data>` returned, if any, depends on the specific AcqG2Ops action that was executed.

G2READ (0X01) COMMAND & RESULTS

The G2Read operation performs a low-level read of any readable portion of tag memory. You have to specify the bank, starting word to be read, and how many words to read back. If you ask the reader to read zero words, the reader will read all the data remaining in the bank (if the tag supports this feature).

```
AcqG2Ops = <opsNum> G2Read <bank> <wordPtr> <wordCount>
AcqG2Ops = <opsNum>   01   <bank> <wordPtr> <wordCount>

// Read the first three words of USER memory
AcqG2Ops = 1 G2Read 3 0 3
```

Note that a locked AccessPwd or KillPwd, or read-locked USER memory, cannot be read unless you provide the correct AcqG2AccessPwd.

The <data> in the AcqG2Ops result is one or more blocks of hex data, formatted according to the TagDataFormatGroupSize setting (0=compressed hex, 1=bytes, 2=words).

```
// Successful G2Read result
01 00 FEED FACE BEEF
```

G2WRITE (0X02) COMMAND & RESULTS

The G2Write operation performs a low-level write to any writeable portion of tag memory. Data that is write-locked can only be written if you supply the correct AcqG2AccessPwd, and data that is PermaLocked cannot be written at all. You specify the bank, starting word pointer to write, followed by the data to be written, and an optional increment symbol.

```
AcqG2Ops = <opsNum> G2Write <bank> <wordPtr> <2n:data…> [+, -, ++]
AcqG2Ops = <opsNum>   02    <bank> <wordPtr> <2n:data…> [+, -, ++]

// Write a kill password
AcqG2Ops = 1 G2Write 0 0 DE AD DE AD

// Write an incrementing EPC
AcqG2Ops = 1 G2Write 1 2 00 00 00 00 00 00 00 00 00 00 +
```

If you choose to auto-increment the data, the AcqG2Ops will continuously keep track of the next value to write, so if you stop and restart the reader's inventories, it will continue on where it left off. You should take care to avoid programming many tags with exactly the same EPC code, since further operations on those tags will be difficult due to the reader's inability to tell them apart.

The <data> in the AcqG2Ops result for a G2Write is just the result code if no incrementing was done, but will contain the actual data written if incrementing is enabled. That written hex data is formatted according to the TagDataFormatGroupSize setting (0=compressed hex, 1=bytes, 2=words). When the G2Write involves incrementing, the command code changes from 0x02 to 0x82.

```
// Successful G2Write result (non-incrementing)
02 00

// Successful G2Write result (incrementing)
82 00 0000 0000 0000 0000 0000 01F3
```

G2KILL (0X03) COMMAND & RESULTS

The G2Kill operation completely disables a tag, preventing it from working in future inventories. In order to kill a tag, it must already have a non-zero Kill Password stored in it, and you must supply the corresponding Kill Password in the kill command itself.

This is a very dangerous command – use with caution, as it will affect all tags in the vicinity!

```
AcqG2Ops = <opsNum> G2Kill <4:killPwd>
AcqG2Ops = <opsNum>   03   <4:killPwd>

// Kill a Tag
AcqG2Ops = 1 G2Kill DE AD DE AD
```

The result of a G2Kill operation is just the success/error code.

```
// Successful G2Kill result
03 00
```

G2LOCK (0X05) COMMAND & RESULTS

The G2Lock command changes the lock status of a field/bank of tag memory. The term "field" is used instead of "bank" because the Reserved bank, bank 0, contains a Kill Password and an Access Password, which can be locked independently. You can Lock or Unlock a field as you wish, but can only PermaLock or PermaUnlock a field once.

A locked field can only be rewritten if you supply the correct AcqG2AccessPwd for the tag. A locked Access Password of Kill Password cannot be read either. The TID bank is always PermaLocked.

You pass as an argument the <field> to be locked, as well as the desired <lockType>.

```
AcqG2Ops = <opsNum> G2Lock <field> <lockType>
AcqG2Ops = <opsNum>   05   <field> <lockType>

// Lock the User Bank
AcqG2Ops = 1 G2Lock 3 Lock

// PermaLock the Kill Password
AcqG2Ops = 1 G2Lock 0 PermaLock
```

The <field> is one of the following integers:

```
 0 = Kill password
10 = Access password
 1 = EPC bank
 2 = TID bank
 3 = USER bank
```

The <lockType> is one of the following strings, or their shortcuts:

```
 L or Lock
 U or Unlock
PL or PermaLock
PU or PermaUnlock
```

The result of a G2Lock operation is just the success/error code.

```
// Successful G2Lock result
05 00
```

G2BLOCKERASE (0X06) COMMAND & RESULTS

The G2BlockErase operation erases (sets to 00) the specified range of tag memory. You specify the bank, starting word to be erased, and how many words to erase. This command is marginally faster than a similar G2Write, because there is no need to transmit all of those bytes filled with zeros.

```
AcqG2Ops = <opsNum> G2BlockErase <bank> <wordPtr> <wordCount>
AcqG2Ops = <opsNum>      06      <bank> <wordPtr> <wordCount>

// Erase the first four words of USER memory
AcqG2Ops = 1 G2BlockErase 3 0 4
```

The result of a G2BlockErase operation is just the success/failure code.

```
// Successful G2BlockErase result
06 00
```

G2BLOCKPERMALOCK (0X08) COMMAND & RESULTS

The G2BlockPermalock operation permanently write-locks specific blocks of user memory. Different tags and different tag manufacturers define just how big a "block" of User memory is. You specify the starting block number, and one or more 16-bit masks indicating the desired PermaLock state of each corresponding block of memory. Most current tags have 16 User blocks or fewer, so typically the <blockPtr> is 0, and there is only one <2:blockMask>.

```
AcqG2Ops = <opsNum> G2BlockPermalock <blockPtr> <2:blockMask1>…
AcqG2Ops = <opsNum>        08        <blockPtr> <2:blockMask1>…

// Permanently lock the first two User blocks
// "C0 00" = 11000000 00000000 → first two blocks locked
AcqG2Ops = 1 G2BlockPermalock 0 C0 00
```

The result of a G2BlockPermalock operation is just the success/failure code.

```
// Successful G2BlockPermalock result
08 00
```

G2GETPERMALOCK (0X09) COMMAND & RESULTS

The G2GetPermalock operation returns the current BlockPermalock status of blocks of User memory. You specify the number of the starting block (in increments of 16), and will get back the 2-byte status of the following 16 blocks.

```
AcqG2Ops = <opsNum> G2GetPermalock <blockPtr>
AcqG2Ops = <opsNum>       09       <blockPtr>

// Fetch the Permalock status of the first blocks of USER memory
AcqG2Ops = 1 G2GetPermalock 0
```

The result of a G2GetPermalock operation is the success/failure code, followed by the <2:blockMask> data containing the status of each block (corresponding to the bits in the word, from most-significant (leftmost) bit, to least-significant (right-most) bit.

```
// Successful G2GetPermalock result
// "C0 00" = 11000000 00000000 → first two blocks locked
09 00 C0 00
```

ALIENG2USERREADLOCK (0X0A) COMMAND & RESULTS

The AlienG2UserReadLock operation utilizes a feature of Alien Higgs-based tags that allows you to hide individual blocks of User memory, by applying a "read-lock" on those blocks. Blocks that are read-locked will return zeros or an error if you attempt to read them without supplying the correct AcqG2AccessPwd. Individual blocks of user memory are addressed similarly to the G2BlockPermalock command, with a <1:blockMask> argument, except this mask is a single byte, rather than a 16-bit word used for G2BlockPermalock. Bits in <1:blockMask> correspond to blocks in User memory, from most-significant bit to least-significant bit.

```
AcqG2Ops = <opsNum> AlienG2UserReadLock <1:blockMask>
AcqG2Ops = <opsNum>                  0A <1:blockMask>

// Hide the first 3rd and 4th blocks of USER memory
// "30" = 00110000 → 3rd and 4th blocks
AcqG2Ops = 1 AlienG2UserReadLock 30
```

The result of an AlienG2UserReadLock operation is just the success/failure code.

```
// Successful AlienG2UserReadLock result
0A 00
```

ALIENG2TAGSTATUS (0X8C) COMMAND & RESULTS

The AlienG2TagStatus operation can return to you the status of all of the write-locks of each of the tag's fields/banks, and the status of the AlienG2UserReadLocks (i.e. "hidden" blocks) in the User bank. The argument to the command is a single hex bitmask, <1:statusMask>, whose value indicates which status elements you would like returned. The AlienG2TagStatus is currently supported only by Alien Higgs4-based tags.

```
AcqG2Ops = <opsNum> AlienG2TagStatus <1:statusMask>
AcqG2Ops = <opsNum>        8C        <1:statusMask>

<1:statusMask>:   1 = bank/field write-locks (bit 1)
                  2 = User read-locks (bit 2)
                  3 = both write-locks and read-locks (bits 1 & 2)

// Ask for both the bank write-locks and the user read-locks
AcqG2Ops = 1 AlienG2TagStatus 03
```

The result of an AlienG2TagStatus operation is the success/failure code, followed by a header word containing the number of status operations being reported and the associated statusMask, followed by a 16-bit word for each status operation, which is a bitmask of field/bank or User block lock states, depending on which was requested.

```
<1:numStatusBits><1:statusMask> [<2:writeLockMask>] [<2:readLockMask>]

Where,
   <writeLockMask> : Lock/Unlock and Permalock bits are given for each tag field,
                     spanning the two bytes:
```

Byte 1:	KPW_l	KPW_p	APW_l	APW_p	EPC_l	EPC_p	TID_l	TID_p
Byte 2:	USR_l	USR_p	RFU	RFU	RFU	RFU	RFU	RFU

```
                     _l: 1=Locked, 0=Unlocked
                     _p: 1=permanent lock, 0=regular lock

Note: An unlocked tag will report 0300, since the TID bank is PermaLocked at the
factory.

   <readLockMask>  : Each bit, starting with the most-significant bit, corresponding
                     to each User block's read-lock status:
```

Byte 1:	block1	block2	block3	block4	block5	block6	block7	block8
Byte 2:	RFU	RFU	RFU	RFU	RFU	RFU	RFU	RFU

Here are some examples of result data from the AlienG2TagStatus operation:

```
// AlienG2TagStatus result – just the write-locks
// "A340" = 10100011 01000000
//    → KillPwd and AccessPwd are locked, EPC unlocked, TID PermaLocked
//    → User PermaUnlocked
8C 00 0101 A340

// AlienG2TagStatus result – write- and read-locks
// write-locks = "A340" (same as above)
// read-locks  = "3000" = 00110000 00000000 → 3rd and 4th blocks read-locked
8C 00 0203 A340 3000
```

Alien *Blast*Write™ – Special Higgs4 Tag Capability

9900 | 9680 | 9650

All of the remaining AcqG2Ops actions involve Alien's new *Blast*Write™ capability, introduced in Higgs4-based tags. *Blast*Write allows the reader to perform the same operation simultaneously, on tags that have been inventoried. Since the reader addresses multiple tags simultaneously, the operations can be applied to the tags very quickly. You can provision entire collections of tags - writing common portions of EPC and USER memory, writing Access and Kill passwords, and locking various fields - all with a single, very fast reader operation.

*Blast*Write is done in conjunction with a tag inventory, except instead of performing the operation one tag at a time as they are inventoried, the reader collects the list of tags seen during the inventory and then performs the *Blast* operation on that entire group of tags at the end. Because of this, when you are configuring your AcqG2Ops operations, you should put the blast-related operations at the end of the sequence of operations.

None of the blast operations may require the tag to be in the secure state (i.e. the tag must not have a non-zero access password). This is because when tags are placed into the special "pingable" state to prepare them for blasting, they drop back to the "arbitrate" state, even if they were previously in the "secure" state (you supplied the correct, non-zero AcqG2AccessPwd). Because of this, writing an access password (which enables the tag security) should be one of the last things you do.

Because of the "bulk nature" of the operation, none of the *Blast*Write operations can return data from the tag.

ALIENG2BLASTLOCK (0X85) COMMAND & RESULTS

The AlienG2BlastLock command changes the lock status of a field/bank of memory for all tags in the field at once. The term "field" is used instead of "bank" because the Reserved bank, bank 0, contains a Kill Password and an Access Password, which can be locked independently. You can Lock or Unlock a field as you wish, but can only PermaLock or PermaUnlock a field once.

You pass as an argument the <field> to be locked, as well as the desired <lockType>.

```
AcqG2Ops = <opsNum> AlienG2BlastLock <field> <lockType>
AcqG2Ops = <opsNum>        85        <field> <lockType>

// Lock the User Banks of all tags
AcqG2Ops = 1 AlienG2BlastLock 3 Lock

// PermaLock the Kill Passwords of all tags
AcqG2Ops = 1 AlienG2BlastLock 0 PermaLock
```

The <field> is one of the following integers:

```
 0 = Kill password
10 = Access password
 1 = EPC bank
 2 = TID bank
 3 = USER bank
```

The <lockType> is one of the following strings, or their shortcuts:

```
 L or Lock
 U or Unlock
PL or PermaLock
PU or PermaUnlock
```

The result of a AlienG2BlastLock operation is just the success/error code.

```
// Successful AlienG2BlastLock result
85 00
```

ALIENG2BLASTERASE (0X86) COMMAND & RESULTS

The AlienG2BlastErase operation erases (sets to 00) the specified range of tag memory for all tags in the field at once. You specify the bank, starting word to be erased, and how many words to erase. This command is marginally faster than a similar G2Write, because there is no need to transmit all of those bytes filled with zeros.

```
AcqG2Ops = <opsNum> AlienG2BlastErase <bank> <wordPtr> <wordCount>
AcqG2Ops = <opsNum>        86         <bank> <wordPtr> <wordCount>

// Erase the first four words of USER memory of all tags
AcqG2Ops = 1 AlienG2BlastErase 3 0 4
```

The result of a AlienG2BlastErase operation is just the success/failure code.

```
// Successful AlienG2BlastErase result
86 00
```

ALIENG2BLASTWRITE (0X87) COMMAND & RESULTS

The AlienG2BlastWrite operation performs a low-level write to any writeable portion of tag for all tags in the field at once. You specify the bank, starting word pointer to write, followed by the data to be written. There is no increment option with AlienG2BlastWrite, since the same command must be issued to all the tags at once.

```
AcqG2Ops = <opsNum> AlienG2BlastWrite <bank> <wordPtr> <2n:data…>
AcqG2Ops = <opsNum>        87         <bank> <wordPtr> <2n:data…>

// Write a kill password
AcqG2Ops = 1 AlienG2BlastWrite 0 0 DE AD DE AD
```

The <data> in the AcqG2Ops result for an AlienG2BlastWrite is just the result code of the operation.

```
// Successful AlienG2BlastWrite result
87 00
```

ALIENG2BLASTBLOCKPERMALOCK (0X88) COMMAND & RESULTS

The AlienG2BlastBlockPermalock operation permanently write-locks specific blocks of user memory for all tags in the field at once. Different tags and different tag manufacturers define just how big a “block” of User memory is. You specify the starting block number, and one or more 16-bit masks indicating the desired PermaLock state of each corresponding block of memory. Most current tags have <= 16 User blocks, so typically the `<blockPtr>` is 0, and there is only one `<2:blockMask>`.

```
AcqG2Ops = <opsNum> AlienG2BlastBlockPermalock <blockPtr> <2:blockMask1>…
AcqG2Ops = <opsNum>             88             <blockPtr> <2:blockMask1>…

// Permanently lock the first two User blocks of all tags
// "C0 00" = 11000000 00000000 → first two blocks locked
AcqG2Ops = 1 AlienG2BlastBlockPermalock 0 C0 00
```

The result of an AlienG2BlastBlockPermalock operation is just the success/failure code.

```
// Successful AlienG2BlastBlockPermalock result
88 00
```

ALIENG2BLASTUSERREADLOCK (0X8A) COMMAND & RESULTS

The AlienG2BlastUserReadLock operation utilizes a feature of Alien Higgs-based tags that allows you to hide individual blocks of User memory, by applying a "read-lock" on those blocks. This *Blast* variant of the command operates on all tags visible in the field at once.

Blocks that are read-locked will return zeros or an error if you attempt to read them without supplying the correct AcqG2AccessPwd. Individual blocks of user memory are addressed similarly to the G2BlockPermalock command, with a `<1:blockMask>` argument, except this mask is a single byte, rather than a 16-bit word used for G2BlockPermalock. Bits in `<1:blockMask>` correspond to blocks in User memory, from most-significant bit to least-significant bit.

```
AcqG2Ops = <opsNum> AlienG2BlastUserReadLock <1:blockMask>
AcqG2Ops = <opsNum>           8A             <1:blockMask>

// Hide the first 3rd and 4th blocks of USER memory of all tags
// "30" = 00110000 → 3rd and 4th blocks
AcqG2Ops = 1 AlienG2BlastUserReadLock 30
```

The result of an AlienG2BlastUserReadLock operation is just the success/failure code.

```
// Successful AlienG2UserReadLock result
8A 00
```

AcqG2OpsMode

9900 | 9680 | 9650

The AcqG2OpsMode command controls whether the configured AcqG2Ops operations will be executed with every tag inventory that the reader performs. When Off (the default value), the AcqG2Ops are only executed when you issue the "to" command. When On, then AcqG2Ops are executed every time the reader acquires tags, whether due to the "t" or "get taglist" commands, or because Automode is running in the background performing inventories.

- Allowed Values: "ON" | "OFF"
- Default Value: "ON"

AcqG2OpsMode Examples	
Command	>AcqG2OpsMode?
Response	AcqG2OpsMode = Off
Command	>AcqG2OpsMode = On
Response	AcqG2OpsMode = On

AcqTime

9900 | 9680 | 9650

The AcqTime command limits the inventory duration and aborts the inventory when it exceeds the specified time. Aborting the inventory does not discard already collected tags. Note that due to certain timing restrictions of the reader and the RFID protocol you should not expect the inventory duration to be exactly the value you specified in the command. The format of the command is as follows:

```
AcqTime = <time>
```

where <time> is the inventory duration in milliseconds (0 to 30000 msec). AcqTime=0 clears the inventory timing restrictions.

- Allowed Values: Integer (0..30000), in milliseconds
- Default Value: 0 (no inventory timing restrictions)

AcqTime Examples	
Command Response	>AcqTime? AcqTime = 0
Command Response	// abort the inventory if it takes longer than 2 seconds >AcqTime = 2000 AcqTime = 2000

SpeedFilter

9900 | 9680 | 9650

The SpeedFilter command allows you to specify which tags are recognized and placed on the TagList, based on their speed. Only those tag reads with computed speeds within the range(s) of the filter are recognized by the reader. You can specify an inclusive range (e.g. between -1m/s and +1m/s), or an exclusive range (e.g. less than -5m/s or greater than +5m/s).

A speed range is defined by two speeds, S1 and S2, and you can specify up to four separate ranges, each separated from the next with a vertical bar (|). Each speed range can be optionally configured to operate only on certain antennas. A tag is reported if its speed matches at least one of the ranges. Speeds have units of meters/sec.

```
SpeedFilter = S1, S2 [ | S3, S4]…
```

You don't really need the commas,

```
SpeedFilter = S1 S2 [ | S3 S4]…
```

If S1 < S2, all tags with speeds inside the range S1-S2 are reported.
If S1 > S2, all tags with speeds outside the range S2-S1 are reported.

You can have different SpeedFilter ranges on separate antennas by including an optional third parameter in each range: an integer bitmap of antennas on which the range is active. The low-order bits of the antenna bitmap correspond to each numbered antenna port on the reader. For instance, to have a speed range be active only on the 1st two antennas, you use an antenna bitmap of "3" (since 3 = 00000011_{binary}): SpeedFilter = 0 999 3 would filter out only those tags on the 1st two antennas that are moving away (their speed is between 0 and 999). When you don't specify an antenna bitmap, the reader assumes you mean "all antennas" and uses a value of 15 (for a 4-port reader) or 3 (for a 2-port reader).

Take note that if you use antenna bitmaps on your filter ranges, that any read coming in on an antenna that doesn't match any of your antenna bitmaps will be ignored. A tag read must pass at least one of the

filters in order to be reported, and if none of the antenna bitmaps match the antenna of the tag read, then the data is thrown away.

```
SpeedFilter = S1 S2 [A1] [ | S3 S4 [A2]]…
```

By default, the tags whose speeds could not be calculated accurately (due to an aborted communication while interrogating the tag or noise) are discarded and not reported in the tag list. The "nospeed" option of the SpeedFilter makes the reader keep those tags on the tag list, but assigns a 'non valid' speed value of 999.00. This way you get to see the tag read anyway, and it is simple to see that the speed value is not valid. The direction for tags with invalid speeds is reported as '0'.

- Turning on SpeedFilter causes the reader to perform extra work while reading tags, which may impact read performance.
- You can see the actual speed of each tag in the TagList be specifying a custom TagList format which includes the %s or ${SPEED} tokens.
- To disable SpeedFilter, simply set it to 0. This is the default value.
- To include tags whose speeds could not be calculated, add the 'nospeed' option.

SpeedFilter Examples	
Command Response	>SpeedFilter? SpeedFilter = 0
Command Response	// read only tags essentially stationary (inclusive range) >SpeedFilter = -0.5 0.5 SpeedFilter = -0.5 0.5
Command Response	// read only tags that are moving (exclusive range) >SpeedFilter = 1 -1 SpeedFilter = 1 -1
Command Response	// read only tags going between -3 and -5 m/s or 0.5 and 1 m/s >SpeedFilter = -5 -3 | 0.5 1 SpeedFilter = -5 -3 | 0.5 1

Command
Response
	// read ALL tags (do not filter based on speed values and
	// also report tags for which speed could not be calculated)
	>SpeedFilter = nospeed
	SpeedFilter = nospeed

Command
Response
	// read tags with speeds between -1 and 1 m/sec as well as
	// tags for which speed could not be calculated
	>SpeedFilter = -1, 1 | nospeed
	SpeedFilter = -1, 1 | nospeed

Command Response	// read only stationary tags on ant #0 and moving tags on ant #1 >SpeedFilter = -0.5 0.5 1 | 1 -1 2 SpeedFilter = -0.5 0.5 1 | 1 -1 2

RSSIFilter

9900 | 9680 | 9650

The RSSIFilter command allows you to see only those tags that have particular received signal strengths. You might only want to read tags that are strong (generally closer) or weak (generally farther). Unlike the SpeedFilter, enabling the RSSIFilter does not require the reader to perform any extra work when reading tags, so read performance should not be impacted.

An RSSI range is defined by two values, R1 and R2, and you can specify up to four separate ranges, each separated from the next with a vertical bar (|). Each RSSI range can be optionally configured to operate only on certain antennas.

A tag is reported if its RSSI matches at least one of the ranges. The values for RSSI reported by the reader are unit-less quantities, and different reader models may report different RSSI values under the same conditions, due to the different radio architectures. Some tuning of the RSSIFilter values may be required on each reader to give consistent results across an enterprise deployment.

```
RSSIFilter = R1, R2 [ | R3, R4]…
```

You don't really need the commas,

```
RSSIFilter = R1 R2 [ | R3 R4]…
```

If R1 < R2, all tags with RSSI inside the range R1-R2 are reported – an inclusive range.
If R1 > R2, all tags with RSSI outside the range R2-R1 are reported – an exclusive range. This is useful to filter tags with a large RSSI, more than, say, 10,000. A range of "10000 0" filters tags with RSSI greater than 10000 or less than 0 (which is impossible).

You can have different RSSIFilter ranges on separate antennas by including an optional third parameter in each range: an integer bitmap of antennas on which the range is active. The low-order bits of the antenna bitmap correspond to each of the numbered antenna ports on the reader. For instance, to have an RSSI range be active only on the 1st two antennas, you would use an antenna bitmap of "3" (since 3 = 00000011_{binary}): RSSIFilter = 5000 0 3 would filter out only those tags on the 1st two antennas that are strong (their RSSI is larger than 5000). When you don't specify an antenna bitmap, the reader assumes you mean "all antennas" and uses a value of 15 (for a 4-port reader) or 3 (for a 2-port reader).

Take note that if you use antenna bitmaps on your filter ranges, that any read coming in on an antenna that doesn't match any of your antenna bitmaps will be ignored. A tag read must pass at least one of the filters in order to be reported, and if none of the antenna bitmaps match the antenna of the tag read, then the data is thrown away.

```
RSSIFilter = R1 R2 [A1] [ | R3 R4 [A2]]…
```

- RSSIFilter does not require the reader to perform extra work, so it shouldn't negatively impact read performance.
- You can see the actual RSSI value of each tag in the TagList be specifying a custom TagList format which includes the %m or ${RSSI} tokens.
- To disable RSSIFilter, simply set it to 0. This is the default value.

RSSIFilter Examples	
Command Response	>RSSIFilter? RSSIFilter = 0
Command Response	// Show only tags with RSSI between 0 and 35000 > RSSIFilter = 0 35000 RSSIFilter = 0 35000
Command Response	// Show only tags with RSSI lower than 5000 or higher than 35000 > RSSIFilter = 35000 5000 RSSIFilter = 35000 5000

Command
Response
	// Show only weak tags on ant #0,1 and string tags on ant #2,3
	// antennas 0 & 1 : 00000011_{binary} = 3 for antenna bitmap
	// antennas 2 & 3 : 00001100_{binary} = 12 for antenna bitmap
	> RSSIFilter = 0 1000 3 | 1000 0 12
	RSSIFilter = 0 1000 3 | 1000 0 12

TagStreamCountFilter

9900 | 9680 | 9650

The TagStreamCountFilter allows you to tell the reader to only stream tag data once a given tag has been read a minimum number of times and/or until it reaches a maximum count.

```
TagStreamCountFilter = min [max]
```

If just the `min` is specified, then the tag is only streamed if the current count for that tag in the reader's taglist is greater than or equal to `min`. If the optional `max` value is specified, then the tag will not be streamed once the count exceeds the `max`. Allowed ranges for both `min` and `max` are 0...65535, and the `max` (if specified) must be greater than the `min`.

The default value is "`0`", which effectively turns off the TagStreamCountFilter and causes the reader to stream all tags. This also happens to have the same effect as "`TagStreamCountFilter = 1`".
To stream tags only once, use `TagStreamCountFilter = 0 1`.

Note that the current tag count that is used to determine if a tag should be streamed or not is dependent on the concept of PersistTime for the taglist. Also, when using TagStreamMode exclusively to get your tag data, you should set a PersistTime other than -1, otherwise the reader's internal taglist will never be purged of data.

TagStreamCountFilter Examples	
Command Response	>TagStreamCountFilter? TagStreamCountFilter = 0
Command Response	*// stream tags only once* >TagStreamCountFilter = 0 1 TagStreamCountFilter = 0 1

TagAuth

9900+ | 9900 | 9680 | 9650

The TagAuth command provides support for the Alien Dynamic Tag Authentication feature available in Alien Higgs3- and Higgs4-based tags. When TagAuth is enabled the reader performs additional verification procedures while reading tags in order to validate that these are authentic Alien tags.

Enabling TagAuth causes the reader to perform extra work while reading tags and may impact read performance. The format of the command is as follows:

```
TagAuth = off | on | h3 | -h3 | h4 | -h4 | * | not
```

Different options can be combined using the '|' separator. The options are as follows:

Option	Reader Authenticates	Reader Reports
off	do not authenticate	all tags
on	all authenticable tag types (Alien Higgs3 & Higgs4)	all tags (including not authenticated and tags not supporting authentication protocol)
h3	Alien Higgs3 tags	authenticated Higgs3 tags only
-h3	Alien Higgs3 tags	tags that failed authentication, i.e. tags that have the manufacturer's ID set to Higgs3 but fail verification test
h4	Alien Higgs4 tags	authenticated Higgs4 tags only
-h4	Alien Higgs4 tags	tags that failed authentication, i.e. tags that have the manufacturer's ID set to Higgs4 but fail verification test
*	Used in conjunction with the h3/-h3/h4/-h4 options to make the reader report all found tags including not-authenticated tags and tags not supporting authentication protocol. This option is useful when TaglistFormat is set to Custom and includes the Tag Authentication token ${AUTH} (see example below).	
not	Used in conjunction with the h3/-h3/h4/-h4 options to invert the meaning of the options. If more than one option is specified using the '|' separator, then the 'not' applies to the entire group of options. The 'not' option cannot be used with 'on' or 'off'.	

The ${AUTH} custom tag list format token is used to report information about tag authentication. The reported value includes tag manufacturer's id code as well as a symbol showing the result of the Alien dynamic tag authentication as follows:

- \+ authentication succeeded
- \- authentication failed

No symbol after the manufacturer's id indicates that either the tag does not support Alien dynamic authentication or the authentication process failed to complete.

If the reader was unable to retrieve manufacturer's id code from the tag (for example, due to noise or an aborted transaction) then the ${AUTH} field is displayed as the '?' symbol.

TagAuth Examples	
Command Response	>TagAuth? TagAuth = OFF
Command Response	*// Dynamically Authenticate Alien Higgs-3 tags and only report Higgs-3 tags* *>TagAuth = h3* TagAuth = h3
Command Response	>TagAuth = h3 | * TagAuth = h3 | * *// Dynamically authenticate Higgs-3 tags but also report ALL other tags* *// found no matter if they passed or failed authentication or whether* *// they support authentication protocol or not.* *// This setting is useful when tag list format is set to 'custom' and* *// includes the Tag Authentication token ${AUTH} as it allows one to see* *// authentication properties as well as tag's manufacturer's codes for* *// all tags in the field.* *// Currently there is only one type of authentication (Higgs-3) supported,* *// so the 'h3 | *' setting happens to have the same effect as the* *// option 'on' (authenticate all known tag types and report all tags)*
Command Response	>TagAuth = -h3 TagAuth = -h3 *// Authenticate Higgs-3 tags and only report tags that **failed** dynamic* *// authentication. This could be an indication that this is a fake Higgs-3 tag.* *// Note that the reader will not report tags not supporting Alien Dynamic* *// Authentication, for example Alien Higgs-2 tags.*
Command Response	>TagAuth = h3 | -h3 TagAuth = h3 | -h3 *// Authenticate Higgs-3 tags and only report Alien Higgs-3 tags* *// that **passed** verification and/or tags that have Alien Higgs-3* *// signature (manufacturer's Id) but **failed** the verification procedure.* *// Other types of tags (for example, Higgs-2) will not be reported.*
Command Response	>TagAuth = not h3 TagAuth = not h3 *// Authenticate Higgs-3 tags and **exclude** tags that pass verification* *// process, i.e. the reader will not report tags that are true Alien* *// Higgs-3 tags but other tags like Alien Higgs-2 or Higgs-3 tags that* *// failed authentication, or tags from other manufacturers will be reported.*
Command Response	// authenticate Alien Higgs-3 or Higgs-4 tags, also report all other tags Alien>TagAuth = h3|h4|* TagAuth = h3|h4|* Alien>TagListCustomFormat = %k auth:${AUTH} TagListCustomFormat = %k auth:${AUTH} Alien>TagListFormat = custom TagListFormat = Custom Alien>t A5A5A5A5A5A5A5A5A5000A auth:E200 3411 // Higgs-2 A5A4A5A5A5ABA5A5A5A50019 auth:E200 3412+ // Higgs-3 authenticated 5068696C69707300AAAAAAAA auth:E200 4001 // not Alien tag A5A4A5A5A5ABA5A5A5A5001E auth:? // unable to retrieve tag information A5A4A5A5A5ABA5A5A5A5002C auth:E200 3414+ // Higgs-4 authenticated A5A4A5A5A5ABA5A5A5A5000C auth:E200 3412- // Higgs-3 authentication FAILED 535410012002300340045005 auth:E200 7240 // not Alien tag 4E585000AC1F3681EC880468 auth:E200 6003 // not Alien tag E2003412DC03011706095651 auth:E200 3412+ // Higgs-3 authenticated 4D32494D3333777755556666 auth:E200 1071 // not Alien tag 48322D3015081509150A150B auth:E200 3411 // Higgs-2 A5A4A5A5A5ABA5A5A5A5002F auth:E200 3412+ // Higgs-3 authenticated

AutoMode Commands

AutoMode is an operation mode that enables hands-free monitoring of tags. Setup requires that you issue a series of configuration commands to the reader. These commands detail how and when to read tags, and then when tags are found, whom to tell. Once configured, the reader can be left to operate on its own.

For a detailed description of the AutoMode system please refer to Chapter 2 of this guide.

AutoMode

9900 | 9680 | 9650

The AutoMode command turns on or off the auto mode.

- Allowed Values: "ON" | "OFF"
- Default Value: "OFF"
- When AutoMode is turned on, the reader records the state of the external outputs. When AutoMode is then turned off, the reader reapplies the saved external output state, unless AutoWaitOutput = -1.

AutoMode Examples	
Command Response	>AutoMode? AutoMode = ON
Command Response	>AutoMode=on AutoMode = ON

AutoWaitOutput

9900 | 9680 | 9650

The AutoWaitOutput specifies the external output settings to apply while in the wait state of AutoMode. The parameter is a bitmap for the four external outputs, where a "1" sets an output to high, and a "0" sets an output to low. Output #1 is specified by bit 0 in the mask, output #2 is specified by bit 1 in the mask, etc.

This is the initial AutoMode state, so setting AutoWaitOutput=0 clears all the outputs each time an AutoMode cycle starts.

- Allowed Values: Integer(-1..255) (max depends on reader model)
- Default Value: 0
- Setting AutoWaitOutput = -1 instructs the reader to not change the external output setting when entering the wait state. Normally, when AutoMode is turned off, it resets the external outputs to the state they were in when AutoMode was turned on. This does not happen if AutoWaitOutput = -1.

AutoWaitOutput Examples	
Command Response	>AutoWaitOutput? AutoWaitOutput = 0
Command Response	>AutoWaitOutput = 3 (sets outputs #1 and #2 high) AutoWaitOutput = 3

AutoStartTrigger

9900 | 9680 | 9650

The AutoStartTrigger specifies the external inputs to monitor to cause the auto mode to jump from wait state to work state. Triggers can either be an input going from low to high (rising edge) or high to low (falling edge). For each type of change, an integer bitmap must be provided to specify the inputs to listen for changes on.

The command takes two space-separated parameters, a rising edge bitmap and a falling edge bitmap. The bitmaps are “OR” bitmaps - if a changing input corresponds to any of the set bits in the bitmap, then AutoMode is triggered. The reader can only listen for rising edges or falling edges at one time, but not both.

You can set the AutoStartTrigger to "-1 -1". This causes the reader to remain in the wait state, regardless of the external inputs. You can then initiate one AutoMode cycle with the AutoModeTriggerNow command. This gives you some manual control (via software) over the Automode state machine.

AutoStartTrigger Examples	
Command Response	>AutoStartTrigger? AutoStartTrigger = 0 0 (rising, falling)
Command Response	>AutoStartTrigger =3 0 AutoStartTrigger = 3 0 (rising, falling)

AutoStartPause

9900 | 9680 | 9650

The AutoStartPause feature allows you to tell the reader to wait a specified number of milliseconds after receiving a start trigger before actually starting.

This allows you to compensate for situations such as a conveyor, where you wish to trigger the reader with a photo-eye, but the photo-eye must be placed some distance from the reader. By setting the AutoStartPause to a value close to the time between when a package crosses the photo-eye's beam and when it reaches the read zone, you eliminate unnecessary RF activity and potential mis-reads from other tags that may happen to still be in the field.

- Allowed Values: Integer(0..86400000)
- Default Value: 0 msec

AutoStartTrigger Examples	
Command Response	>AutoStartPause? AutoStartPause = 0
Command Response	>AutoStartPause = 150 AutoStartPause = 150

AutoWorkOutput

9900 | 9680 | 9650

The AutoWorkOutput specifies the output settings to effect while in the work state of AutoMode. The parameter is a bitmap for the four external outputs, where a "1" sets an output to high, and a "0" sets an output to low. Output #1 is specified by bit 0 in the mask, output #2 is specified by bit 1 in the mask, etc.

- Allowed Values: Integer(-1..255) (max depends on reader model)
- Default Value: 0
- Setting AutoWorkOutput = -1 instructs the reader to not change the external output setting when entering the work state.

AutoWorkOutput Examples	
Command Response	>AutoWorkOutput? AutoWorkOutput =0
Command Response	>AutoWorkOutput =3 (sets outputs #1 and #2 high) AutoWorkOutput =3

AutoAction

9900 | 9680 | 9650

The AutoAction command specifies the action to perform when running in the work state of AutoMode. The reader continues to perform the AutoAction for the duration of the work state. If the AutoAction is a programming command, the reader attempts the operation until it succeeds, or the work state is finished (because of a timer or external trigger).

AutoAction	Description
None	Perform no action
Acquire	Perform tag inventories. This is the default value.
ProgramEPC ("Program")	Programs a tag's EPC. See the Tag Programming chapter for more information.
Erase	Erases a tag's EPC. See the Tag Programming chapter for more information.
ProgramAndLockEPC ("Program and Lock")	Programs and locks a tag's EPC. See the Tag Programming chapter for more information.
ProgramUser	Program' a tag's User memory. See the Tag Programming chapter for more information.
ProgramAndLockUser	Programs and locks a tag's User memory. See the Tag Programming chapter for more information.
ProgramAlienImage	Writes an entire tag image to an Alien Higgs2 or Higgs3 tag.
Macro <macroname>	Run the named reader macro.

AutoAction Examples	
Command Response	>AutoAction? AutoAction = Acquire
Command Response	>AutoAction = Acquire AutoAction = Acquire

AutoStopTrigger

9900 | 9680 | 9650

The AutoStopTrigger specifies the external inputs to monitor to cause the auto mode to jump from work state to evaluate state. Triggers can either be an input going from low to high (rising edge) to high to low (falling edge). For each type of change, an integer bitmap must be provided to specify the inputs to listen for changes on.

The command takes two space-separated parameters, a rising edge bitmap and a falling edge bitmap. The bitmaps are "OR" bitmaps - if a changing input corresponds to any of the set bits in the bitmap, then AutoMode is stopped. The reader can only listen for rising edges or falling edges at one time, but not both.

AutoStopTrigger Examples	
Command Response	>AutoStopTrigger? AutoStopTrigger = 0 0 (rising, falling)
Command Response	>AutoStopTrigger = 3 0 AutoStopTrigger = 3 0 (rising, falling)

AutoStopTimer

9900 | 9680 | 9650

The AutoStopTimer offers an alternative way to jump from work state to evaluate state. This is a time-based stop trigger, which puts an upper-limit on how long the AutoAction runs for in each cycle. The parameter is a single time period, specified in milliseconds.

- Allowed Values: Integer(-1..86400000)
- Default Value: 1000 msec
- AutoStopTimer = -1 indicates the reader should remain in the work state indefinitely, or until a configured AutoStopTrigger is received. If there is no AutoStopTimer or AutoStopTrigger, the reader will never transition into the evaluate/notify state, so the outputs will never change and no notification messages will be sent.
- AutoStopTimer = 0 indicates the reader should perform exactly one operation (read, program, etc.) in its work state, then proceed to the evaluate state.
- An AutoStopTimer of 86,400,000 msec corresponds to 24 hours.

If the reader is in the middle of an inventory when the AutoStopTimer expires, the inventory completes before moving on to the evaluate state.

AutoStopTimer Examples	
Command Response	>AutoStopTimer? AutoStopTimer = 1000
Command Response	>AutoStopTimer = 5000 AutoStopTimer = 5000

AutoStopPause

9900 | 9680 | 9650

The AutoStopPause is a feature which allows you to tell the reader to wait a specified number of milliseconds after receiving a stop trigger before actually stopping. This delay only applies when the AutoAction is stopped because of an external I/O trigger, not because of the AutoStopTimer expiring.

AutoStopPause allows you to compensate for situations such as a conveyor, where you wish to trigger the reader to start and stop with photo-eyes, but the "stop" photo-eye must be placed at a position too close to the reader. By setting the AutoStopPause to a value close to the time between when a package crosses the "stop" photo-eye's beam and when it actually leaves the effective read zone, you allow the reader to continue to do useful work even after the stop trigger has fired.

- Allowed Values: Integer(0..86400000)
- Default Value: 0 msec

AutoStopPause Examples	
Command Response	>AutoStopPause? AutoStartPause = 0
Command Response	>AutoStartPause = 150 AutoStartPause = 150

AutoTrueOutput

9900 | 9680 | 9650

The AutoTrueOutput specifies the output settings to effect if the evaluate mode of AutoMode evaluates to true. The parameter is a bitmap for the four external outputs, where a "1" sets an output to high, and a "0" sets an output to low. Output #1 is specified by bit 0 in the mask, output #2 is specified by bit 1 in the mask, etc.

- Allowed Values: Integer(-1..255) (max depends on reader model)
- Default Value: 0
- Setting AutoTrueOutput = -1 instructs the reader to not change the external output setting when entering the evaluate true state.

AutoTrueOutput Examples	
Command Response	>AutoTrueOutput? AutoTrueOutput = 0
Command Response	>AutoTrueOutput = 3 (sets outputs #1 and #2 high) AutoTrueOutput = 3

AutoTruePause

9900 | 9680 | 9650

The AutoTruePause specifies a millisecond pause to effect if the autonomous evaluation mode evaluates to true. This pause occurs after the AutoTrueOutput command has been processed. Note that while the reader is pausing, it is not reading tags, which can affect performance.

- Allowed Values: Integer(0..86400000)
- Default Value: 0 msec

AutoTruePause Examples	
Command Response	>AutoTruePause? AutoTruePause(ms) = 0
Command Response	>AutoTruePause = 500 AutoTruePause = 500

AutoFalseOutput

9900 | 9680 | 9650

The AutoFalseOutput specifies the output settings to effect if the evaluate mode of AutoMode evaluates to false. The parameter is a bitmap for the four external outputs, where a "1" sets an output to high, and a "0" sets an output to low. Output#1 is specified by bit 0 in the mask, output #2 is specified by bit 1 in the mask, etc.

- Allowed Values: Integer(-1..255) (max depends on reader model)
- Default Value: 0
- Setting AutoFalseOutput = -1 instructs the reader to not change the external output setting when entering the evaluate false state.

AutoFalseOutput Examples	
Command Response	>AutoFalseOutput? AutoFalseOutput = 0
Command Response	>AutoFalseOutput = 3 AutoFalseOutput = 3

AutoFalsePause

9900 | 9680 | 9650

The AutoFalsePause specifies a millisecond pause to effect if the autonomous evaluation mode evaluates to false. This pause occurs after the AutoFalseOutput command has been processed. Note that while the reader is pausing, it is not reading tags, which can affect performance.

- Allowed Values: Integer(0..86400000)
- Default Value: 0 msec

AutoFalsePause Examples	
Command Response	>AutoFalsePause? AutoFalsePause (ms) = 0
Command Response	>AutoFalsePause = 500 AutoFalsePause = 500

AutoErrorOutput

9900 | 9680 | 9650

The AutoErrorOutput specifies the output settings to apply if the AutoMode action evaluates to false with one of the listed errors. If there is no match then the output is set to the AutoFalseOutput value. The first parameter is a bitmap of external outputs to energize, followed by an optional list of error codes. A bit value of "1" sets an output to high, and a "0" sets an output to low. Output #1 is specified by first (low order) bit of the bitmap, output #2 is specified by the second bit, etc.

The command format is as follows:

```
AutoErrorOutput = <output>[,err1 [,err2]...]
```

If only a single <output> parameter is specified, it must be '-1' (disabled). This is the default value.

- Allowed Values:
  <output bitmap>: Integer (-1..255) (maximum depends on reader type)
  <err>: Integer (1..255)
- Default Value: -1 (disabled)
- Setting AutoFalseOutput = -1 instructs the reader to not change the external output setting when entering the evaluate false state.

AutoErrorOutput Examples	
Command	>AutoErrorOutput?
Response	AutoErrorOutput = -1
Command	// set output pin 2 if AutoMode action results in errors 134 or 149
Response	>AutoErrorOutput = 4 134 149 AutoErrorOutput = 4 134 149

AutoProgError

9900 | 9680 | 9650

When an AutoMode programming action fails, a new tag is added to the tag list with the antenna value set to 255. The AutoProgError command allows you to choose a custom error value or report the actual error code. Setting AutoProgError to '-1' results in reporting the actual error code. The format is as follows:

```
AutoProgError = <error>
```

- Allowed Values: Integer(-1..255)
- Default Value: 255
- Setting AutoProgError = -1 results in reporting the actual error code.

AutoProgError Examples	
Command	>AutoProgError?
Response	AutoProgError = 255
Command	// set to -1 to report the actual error code
Response	>AutoProgError = -1 AutoProgError = -1

AutoModeReset

9900 | 9680 | 9650

The AutoModeReset command resets all auto mode parameters to their default values, including setting the auto mode to off.

AutoModeReset Examples	
Command	>AutoModeReset
Response	All AutoMode settings have been reset!

AutoModeTriggerNow

9900 | 9680 | 9650

The AutoModeTriggerNow command emulates an external IO trigger event to trigger AutoMode to start. This command only works if the RFID Reader is already in AutoMode and is waiting for a start trigger

condition. At this point, issuing the AutoModeTriggerNow command is identical to a real external trigger event, forcing AutoMode into its action cycle.

AutoModeTriggerNow Examples	
Command	>AutoModeTriggerNow
Response	Auto Mode Triggering Now

Notify Mode Commands

The Notify Mode commands are used to set up automated event notification either upon the expiration of a timer, or triggered off of events that occur while the reader is running in AutoMode.

NotifyMode

9900 | 9680 | 9650

The NotifyMode command turns on or off the notify mode.

- Allowed Values: "ON" | "OFF"
- Default Value: "OFF"

NotifyMode Examples	
Command Response	>NotifyMode? NotifyMode = ON
Command Response	>NotifyMode = on NotifyMode = ON

NotifyAddress

9900 | 9680 | 9650

The NotifyAddress command specifies where notification messages should be sent when they are triggered. The form of the address determines the method of delivery.

There are four delivery methods supported as shown in the table below:

NotifyAddress	Description
hostname:port	Send a message to a specified port on a networked machine. The address takes the form "hostname:port." For example, "123.01.02.98:3450" or "listener.alientechnology.com:10002"
user@domain.com	Send a message via e-mail to the address specified. The address is specified in standard email form, i.e., user@domain.com *NOTE: the MailServer parameter must be configured for this to work. Optionally the MailFrom parameter can be used. No email authentication protocols are supported at this time.*
http://domain:port/path	Send a message as a POST request to a web service. The address must start with "http://", and may optionally include the port number (default is 80) and path to a specific web service. The notification message fills the body of the content part of the POSTed message, with no key=value form-style formatting – you just read the input stream directly.
SERIAL	Send a message to the serial connection. The word "SERIAL" is used as the address. The word is not case sensitive.

You can specify a fail-over NotifyAddress, to be used when a notification attempt to the primary NotifyAddress fails. The reader always tries the primary address first before failing over to the second address. The two addresses may indicate different delivery protocols – the primary might be a TCP socket address, and the fail-over might be an e-mail address, for instance.

You specify the fail-over address after the primary address, separating them with a vertical bar (|):

```
NotifyAddress = <address1> | <address2>
```

NotifyAddress Examples	
Command Response	>NotifyAddress? NotifyAddress = 10.1.0.12:4000
Command Response	>NotifyAddress = user@msn.com NotifyAddress = user@msn.com
Command Response	// Specify a fail-over address >NotifyAddress = 10.1.0.12:4000 | user@msn.com NotifyAddress = 10.1.0.12:4000|user@msn.com

NotifyTime

9900 | 9680 | 9650

The NotifyTime command assigns and retrieves the time interval for automatic TagList pushing to a listening machine.

- Allowed Values: Integer(0..65535)
- Default Value: 0 sec
- The time is specified in seconds.
- If set to zero, the time-based automatic notification is disabled.
- When set to a positive number of seconds, a standard notification message is sent out each period.

NotifyTime Examples	
Command Response	>NotifyTime? NotifyTime = 30
Command Response	>NotifyTime = 30 NotifyTime = 30

NotifyTrigger

9900 | 9680 | 9650

The NotifyTrigger command specifies and retrieves the event conditions (other than time-based) upon which a notification message is sent out. Notify messages can be triggered under any of the following conditions:

Trigger	Meaning
Add	Send message when new tag is read and added to the TagList. Sends only the added tags.
Remove	Send message when a tag is removed from the TagList. Sends only the removed tags.
AddRemove	Send message when a tag is either added to or removed from the TagList. Sends the list of added tags and the list of removed tags.
Change	Send message when a tag is either added to or removed from the TagList. Sends the current TagList
True	Send message when the evaluation task of the autonomous state loop evaluates to true, typically when tags are added to the TagList. Sends the current TagList
False	Send message when the evaluation task of the autonomous state loop evaluates to false, typically when tags are not added to the TagList. Sends the current TagList
TrueFalse	Send message when the evaluation task of the autonomous state loop evaluates to true or false (i.e. every time). Sends the current TagList

NotifyTrigger Examples	
Command Response	>NotifyTrigger? NotifyTrigger = Remove
Command Response	>NotifyTrigger = add NotifyTrigger = Add

NotifyFormat

9900 | 9680 | 9650

The NotifyFormat parameter specifies the format of any notification message. The format may be one of the following:

Format	Description
Text	TagLists are sent out as plain text messages, one tag ID per line.
Terse	Similar to the Text format, except TagList data is formatted in the terse format.
XML	TagLists are sent out in XML text format.
Custom	Similar to the Text format, except TagList data is formatted as defined by the TagListCustomFormat command.

- Text-formatted TagLists take the form:

```
#Alien RFID Reader Auto Notification Message
#ReaderName: Spinner Reader
#ReaderType: Alien RFID Tag Reader (Class 1 / 915MHz)
#IPAddress: 10.1.70.13
#CommandPort: 23
#MACAddress: 00:80:66:10:11:6A
#Time: 2003/01/21 12:48:59
#Reason: TEST MESSAGE
Tag:8000 8004 0000 003B, Disc:2003/12/04 15:08:59, Last:2003/12/04 15:08:59, Count:4,
Ant:0
Tag:8000 8004 9999 0004, Disc:2003/12/04 15:08:59, Last:2003/12/04 15:08:59, Count:3,
Ant:0
#End of Notification Message
```

- Terse-formatted TagLists take the form:

```
#Alien RFID Reader Auto Notification Message
#ReaderName: Spinner Reader
#ReaderType: Alien RFID Tag Reader (Class 1 / 915MHz)
#IPAddress: 10.1.70.13
#CommandPort: 23
#MACAddress: 00:80:66:10:11:6A
#Time: 2003/01/21 12:48:59
#Reason: TEST MESSAGE
8000 8004 0000 003B,0,4
8000 8004 9999 0004,0,3
#End of Notification Message
```

- XML-formatted TagLists take the form:

```
<Alien-RFID-Reader-Auto-Notification>
  <ReaderName>Spinner Reader</ReaderName>
  <ReaderType><Alien RFID Tag Reader (Class 1 / 915MHz)</ReaderType>
  <IPAddress>10.1.70.13</IPAddress>
  <CommandPort>23</CommandPort>
  <MACAddress>00:80:66:10:11:6A</MACAddress>
  <Time>2003/01/21 12:49:22</Time>
  <Reason>TEST MESSAGE</Reason>
    <Alien-RFID-Tag-List>
        <Alien-RFID-Tag>
        <TagID>8000 8004 0000 003B </TagID>
        <DiscoveryTime>2003/12/04 15:08:59</DiscoveryTime>
        <LastSeenTime>2003/12/04 15:08:59</LastSeenTime>
          <Antenna>0</Antenna>
        <ReadCount>4</ReadCount>
          <Protocol>1</Protocol>
        </Alien-RFID-Tag>
        <Alien-RFID-Tag>
        <TagID>8000 8004 9999 0004</TagID>
        <DiscoveryTime>2003/12/04 15:08:59</DiscoveryTime>
        <LastSeenTime>2003/12/04 15:08:59</LastSeenTime>
        <Antenna>0</Antenna>
        <ReadCount>3</ReadCount>
          <Protocol>1</Protocol>
        </Alien-RFID-Tag>
    </Alien-RFID-Tag-List>
</Alien-RFID-Reader-Auto-Notification>
```

NotifyHeader

9900 | 9680 | 9650

The NotifyHeader command turns on or off the header portion of each notification message. When the NotifyHeader is turned off, notification messages contain only the TagList portion of the message.

- Allowed Values: "ON" | "OFF"
- Default Value: "ON"

NotifyMode Examples	
Command Response	>NotifyHeader? NotifyHeader = On
Command Response	>NotifyHeader = off NotifyHeader = Off

NotifyKeepAliveTime

9900 | 9680 | 9650

When the reader sends a notification message out over the network, it needs an open TCP socket to do so. Opening and closing a socket entails some processor and network overhead, and if the notifications are coming out of the reader rapidly, this overhead can become a burden that hinders reader responsiveness.

The NotifyKeepAliveTime attribute sets the amount of time (in seconds) that the reader keeps this socket open. For rapid notifications, it may be advantageous for the reader to hold the socket open continuously. You do this by setting NotifyKeepAliveTime to some value greater than the expected time between notifications.

On the other hand, holding the socket open places a burden on the network. For infrequent notifications, it is advantageous to leave the NotifyKeepAliveTime small so that the socket is opened only long enough for a single notification to be sent out, and is then closed automatically after message delivery.

- Allowed Values: Integer(0..65535)
- Default Value: 30

NotifyKeepAliveTime Examples	
Command Response	>NotifyKeepAliveTime? NotifyKeepAliveTime = 30
Command Response	>NotifyKeepAliveTime = 90 NotifyKeepAliveTime = 90

MailServer

9900 | 9680 | 9650

The MailServer command allows you to define an SMTP (simple mail transfer protocol) mail server. The SMTP server must support non-authenticated access. This mail server is used only when automatic notification is configured (see Notify commands) and is set to use Mail as its delivery method.

MailServer Examples	
Command Response	>MailServer? MailServer = 12.34.56.78
Command Response	>MailServer = 45.224.124.34 MailServer = 45.224.124.34

MailFrom

9900 | 9680 | 9650

The MailFrom command allows you to define the email address associated with the RFID Reader. The emails sent out by the RFID Reader have this parameter set in the From: field of the email header.

MailFrom Examples	
Command Response	>MailFrom? MailFrom = AlienRFIDReader
Command Response	>MailFrom = reader@mycompany.com MailFrom = reader@mycompany.com

NotifyRetryCount

9900 | 9680 | 9650

If a network notification fails to connect to a host at the specified NotifyAddress, the reader attempts to resend the notification message. Rather than endlessly retrying a failed operation (which needlessly consumes reader resources), the reader stops after the number of attempts reaches the NotifyRetryCount. At this point the reader turns NotifyMode off.

Readers can queue up sequences of failed notification messages (up to 1000 messages), and they are all delivered once connectivity is reestablished, either after you resolve the underlying network problem, or by configuring a different NotifyAddress. This feature obviates the need for a NotifyMode "kill switch", so the NotifyRetryCount default value is –1 (never stop retrying) in these readers.

- Allowed Values: Integer(-1..32767)
- Default Value: -1 (never give up retrying)
- The period between retries is given by the NotifyRetryPause (below).

NotifyRetryCount Examples	
Command Response	>NotifyRetryCount? NotifyRetryCount = 3
Command Response	>NotifyRetryCount = 0 NotifyRetryCount = 0

NotifyRetryPause

9900 | 9680 | 9650

When the reader attempts to send a network notification and fails, it tries again a number of times specified by NotifyRetryCount. The time period between these retries is specified by the NotifyRetryPause.

- Allowed Values: Integer(0..32767)
- Default Value: 10 sec

NotifyRetryPause Examples	
Command Response	>NotifyRetryPause? NotifyRetryPause = 10
Command Response	>NotifyRetryPause = 60 NotifyRetryPause = 60

NotifyQueueLimit

9900 | 9680 | 9650

The NotifyQueueLimit command allows you to specify how many failed notification messages the reader should queue for later delivery. When notifying to a TCP socket or mail server, failed deliveries are queued up inside the reader and retried at period intervals (specified by NotifyRetryPause). Up to 1000 notifications can be queued, and once the queue limit is reached, older messages are dropped in favor of newer messages.

Setting NotifyQueueLimit to smaller values decreases the maximum number of messages in the queue, and therefore how many "stale" messages are delivered once the notification channel is reestablished. Setting NotifyQueueLimit to zero prevents any failed messages from being resent.

- Allowed Values: Integer(0..1000)
- Default Value: 1000

NotifyQueueLimit Examples	
Command Response	>NotifyQueueLimit? NotifyQueueLimit = 1000
Command Response	>NotifyQueueLimit = 0 NotifyQueueLimit = 0

NotifyInclude

9900 | 9680 | 9650

The NotifyInclude command tells the NotifyMode engine whether to include the current IOList in notification messages instead of, or in addition to, the normal TagList. It may have one of the following values:

NotifyInclude	Description
Tags	Notification messages only include the TagList. This is equivalent to the traditional notification messages, and is the default setting.
DI	Notification messages only include digital input events.
DO	Notification messages only include digital output events.
DIO	Notification messages only include both digital input and digital output events (interleaved).
All	Notification messages include the standard TagList, as well as all digital I/O events.

- The default value is "Tags".
- When notification messages include both tag and I/O data, the tag data always comes first.
- The formatting of both the TagList and IOList are specified with the NotifyFormat command.
- Following is sample data, with NotifyFormat=Text, and NotifyInclude=All. Note that there is no explicit division between the TagList and IOList:

```
#Alien RFID Reader Auto Notification Message
#ReaderName: Alien RFID Reader
#ReaderType: Alien RFID Tag Reader, Model: ALR-9800 (Four Antenna / Multi-Protocol / 902-
928 MHz)
#IPAddress: 10.10.82.72
#CommandPort: 23
#MACAddress: 00:80:66:10:11:6A
#Time: 2007/01/22 11:25:45
#Reason: TEST MESSAGE
#StartTriggerLines: 0
#StopTriggerLines: 0
Tag: E200 3411 B801 0108 1209 0054, Disc:2007/01/22 11:25:22, Last:2007/01/22 11:25:43,
Count:20, Ant:0, Proto:2
IO:DI, Time:2007/01/22 11:25:22.343, Data:1
IO:DO, Time:2007/01/22 11:25:22.361, Data:2
#End of Notification Message
```

- The same data, with NotifyFormat=XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<Alien-RFID-Reader-Auto-Notification>
  <ReaderName>Alien RFID Reader</ReaderName>
  <ReaderType>Alien RFID Tag Reader, Model: ALR-9900 (Four Antenna / Multi-Protocol /
902-928 MHz)</ReaderType>
  <IPAddress>10.10.82.72</IPAddress>
  <CommandPort>23</CommandPort>
  <MACAddress>00:80:66:10:11:6A</MACAddress>
  <Time>2007/01/22 11:25:45</Time>
  <Reason>TEST MESSAGE</Reason>
  <StartTriggerLines>0</StartTriggerLines>
  <StopTriggerLines>0</StopTriggerLines>
    <Alien-RFID-Tag-List>
        <Alien-RFID-Tag>
        <TagID>E200 3411 B801 0108 1209 00546</TagID>
        <DiscoveryTime>2007/01/22 11:25:22</DiscoveryTime>
        <LastSeenTime>2007/01/22 11:25:22</LastSeenTime>
        <Antenna>0</Antenna>
        <ReadCount>2</ReadCount>
        <Protocol>2</Protocol>
        </Alien-RFID-Tag>
    </Alien-RFID-Tag-List>
    <Alien-RFID-IO-List>
        <Alien-RFID-IO>
        <Type>DI</Type>
        <Time>2007/01/22 11:25:22.343</Time>
        <Data>1</Data>
        </Alien-RFID-IO>
        <Alien-RFID-IO>
         <Type>DO</Type>
         <Time>2007/01/22 11:25:22.361</Time>
         <Data>2</Data>
        </Alien-RFID-IO>
    </Alien-RFID-IO-List>
</Alien-RFID-Reader-Auto-Notification>
```

NotifyInclude Examples	
Command	>NotifyInclude?
Response	NotifyInclude = Tags
Command	>NotifyInclude = All
Response	NotifyInclude = All

NotifyNow

9900 | 9680 | 9650

The NotifyNow command instructs the reader to send out an immediate notification of its TagList to the address currently set by the NotifyAddress command.

NotifyNow Examples	
Command	>NotifyNow
Response	Issuing Notify Trigger...

CHAPTER 5
Tag Programming

All Alien RFID tags support programmable ID numbers. This chapter describes the series of commands required to program EPC-compliant ID codes and user-defined ID codes into Alien RFID tags, and details some of the physical conditions required to carry out successful programming tasks.

Tag Memory Structure

Knowing the tag memory structure is essential for successfully programming tags and using mask commands to acquire subsets of tags. This chapter describes the basic memory formats supported by all Alien tag systems.

Class 1/Gen 2 Tag Memory

The basic size of a data entity in Class1, Gen2 is a word (16 bits), rather than a byte (8 bits). Class 1/Gen 2 tags contain up to four memory banks.

G2 Bank	Description
Bank 0 RESERVED	Contains the Kill and Access passwords. Each is two words (4 bytes) in length. Tags need not implement either of these passwords, in which case the values are taken to be 0000 0000, and the corresponding unused memory locations need not exist. Masking into Bank 0 is not permitted.
Bank 1 EPC	Contains one CRC word, one PC word, and a flexible number of EPC words following. See below for detailed structure.
Bank 2 TID	Contains tag identifying information, such as the allocation class identifier (EPCglobal, or other), manufacturer information, and tag model and versioning information.
Bank 3 USER	Unstructured scratch space for user-specific data storage. Tags need not implement the USER bank, and the memory capacity depends on the tag implementation.

CLASS 1, GEN 2 STRUCTURE

Current Class 1/Gen 2 tags generally contain 96 or 128 bits of programmable memory in the EPC memory bank, although the specification allows for up to 496 EPC bits. In addition to the EPC code, the EPC memory bank also contains 16 bits of CRC checksum, and 16 Protocol-Control (PC) bits.

	CRC	PC	EPC Code (or User ID Code)						
Word	0	1	0	1	2	3	4	5	…
Bit	0-15	16-31	32-47	48-63	64-79	80-95	96-111	112-127	…

128-bit Class 1, Gen 2 Tag Memory Structure

The EPC Code is addressed from left to right, where the leftmost bit (the Most Significant Bit) is bit 32, and extending out through the length of the EPC. There is no restriction on the data that resides in this portion of the tag.

The checksum is automatically calculated by the tag, over the 16 bits of the PC and the entire EPC Code. The checksum is calculated and programmed into the tag automatically by the reader.

The Protocol-Control (PC) word encodes the number of the EPC words, (bits 0-4), followed by two reserved bits (bit 5-6), and followed by a numbering system identifier (bits 7-15). The numbering system identifier specifies either an EPCglobal™ header (defined by the EPC™ Tag Data Standards) or an Application Family Identifier (defined by ISO/IEC 15961).

Programming Distance & Power Levels

Caution: It is highly recommended that you read and understand this section before programming tags.

The term “programming” as used in this manual refers to the operations that alter the tag memory. These include the Erase, ProgramEPC, Lock and Kill commands. These commands are discussed in some detail later in this chapter. From the operational and software point of view, programming tag IDs is very simple, in most cases requiring just the click of a button or the issuance of one command. However, several variables affect programming reliability and must be properly addressed in every application. There are three factors under your control to assure programming success in any application: application software, attenuation, and the physical position of the tags.

Programming Power

Programming a tag requires substantially more power than reading a tag. As a result, the tag's programming range is substantially less than its read range. Programming commands affect all tags that receive sufficient power to execute the commands. As a result, the tag to be programmed should be physically isolated from tags that you do not intend to program, and a mask should be set to match only that one tag. Similarly, you should program at the minimum power that reliably programs the tag in the given environment, so as not to affect nearby tags. A lower programming power requires a shorter physical separation of tags.

The power received by the tag is determined by the power supplied to the antenna, the distance tag is from the antenna, and the level of signal reflections from the environment and the object being tagged. Environmental reflections can cause local power nulls near the tag, and it will not be programmed due to insufficient power. Tag separation from the antenna should be as constant as possible, to minimize power variation and to avoid special nulls due to reflection. Tag orientation with respect to the antenna should also be controlled. In particular, when using a linear antenna the tag should be presented in the same orientation as the antenna polarization. The power supplied to the antenna is controlled by the use of attenuators to reduce the signal strength. The attenuator value is selected based on careful measurements in the environment after other variables have been controlled as described above. A variable attenuator is a useful tool in setting the final attenuation values.

Programming Problems

The most common result of poor programming is that the tag no longer reads in Global Scroll or Inventory modes. This is due to the tag memory being erased or incompletely programmed. Tags in this state can be read using the Verify Tag command and can be programmed using the Program Tag commands.

Typically, tag erasure without programming success (the Verify Tag command returns all 0’s) is caused by insufficient power since the erase process requires less energy than the programming process. Power should be increased by either decreasing attenuation or moving the tag closer to the antenna and repeating the programming process.

Incompletely programmed tags can be caused by insufficient or excessive programming power. Reevaluate the tag position and the signal attenuation and repeat the programming sequence.

A step attenuator is a powerful tool in evaluating programming conditions. When using the attenuator, set it to the highest value at which you can reliably read the tag. You can then step down attenuation (raise

the power) approximately 10 dB from this value as an estimate of the proper attenuation for programming. By varying the attenuator value you can optimize programming conditions over the range of the other variables in your application.

Programming Commands Summary

Command	Description	9900	9680	9650
ProgProtocol	The single protocol (0, 1, 2) used in subsequent programming operations.	✓	✓	✓
ProgAntenna	The antenna on which to issue programming commands.	✓	✓	✓
ProgramEPC ("p")	Writes an EPC code to the tag.	✓	✓	✓
ProgramAndLockEPC	Writes and then locks the EPC bank.	✓		
ProgramAccessPwd	Writes the Gen2 Access password.	✓	✓	✓
ProgramKillPwd	Writes the Gen2 Kill password.	✓	✓	✓
ProgramUser	Writes user data to the Gen2 tag.	✓	✓	✓
ProgramAndLockUser	Writes and then locks the User bank.	✓		
LockEPC	Locks a tag's EPC bank.	✓	✓	✓
LockAccessPwd	Locks a Gen2 tag's Access password.	✓	✓	✓
LockKillPwd	Locks a Gen2 tag's Kill password.	✓	✓	✓
LockUser	Locks a Gen2 tag's User memory.	✓	✓	✓
LockUserBlocks	Perma-lock individual Gen2 User memory blocks	✓		
HideAlienUserBlocks	Hide/Reveal individual User memory blocks (Higgs3 & Higgs4 tags only).	✓		
UnlockEPC	Unlocks a Gen2 tag's EPC bank.	✓	✓	✓
UnlockAccessPwd	Unlocks a Gen2 tag's Access password.	✓	✓	✓
UnlockKillPwd	Unlocks a Gen2 tag's Kill password.	✓	✓	✓
UnlockUser	Unlocks a Gen2 tag's User memory.	✓	✓	✓
ProgG2LockType	The lock type (Lock, PermaLock, PermaUnlock) used to lock Gen2 tags.	✓	✓	✓
ProgEPCData	The next EPC used to automatically program tags. Was called "ProgramID".	✓	✓	✓
ProgG2NSI	The value that the NSI part of the PC word will be set to when programming with the ProgramEPC command.	✓	✓	✓
ProgG2AccessPwd	The Access password data used to automatically write Gen2 Access passwords.	✓	✓	✓
ProgG2KillPwd	The Kill password data used to automatically write Gen2 Kill passwords.	✓	✓	✓
ProgUserData	The User data used to automatically program a Gen2 tag's User memory.	✓	✓	✓
ProgEPCDataInc	Which EPC programming results cause ProgEPCData to be automatically incremented.	✓	✓	✓
ProgUserDataInc	Which User programming results cause ProgUserData to be automatically incremented.	✓	✓	✓
ProgEPCDataIncCount	Limits the number of ProgEPCData increments.	✓		
ProgUserDataIncCount	Limits the number of ProgUserData increments.	✓		

Command	Description	9900	9680	9650
G2Erase	Erases a specified portion of Gen2 tag memory	✓	✓	✓
Erase	Erases a tag (writes zeros to EPC).	✓	✓	✓
Kill	Kills a tag with a specified kill password.	✓	✓	✓
ProgAttempts	The number of attempts to program a tag.	✓	✓	✓
ProgSuccessFormat	Determines the result format of a successful program operation.	✓	✓	✓
ProgSingulate	Enables/disables a singulation check before performing a programming operation.	✓		
TagInfo	Fetch information about a tag.	✓	✓	✓
G2Read	Perform a low-level read of Gen2 tag memory.	✓	✓	✓
G2Write	Perform a low-level write to Gen2 tag memory.	✓	✓	✓
ProgDataUnit	Specifies the data unit size when writing Gen2 tags.	✓	✓	✓
ProgBlockSize	The size, in words, of data blocks when writing in block mode.	✓		
ProgBlockAlign	Get assistance when performing block writes across block boundaries.	✓		
ProgramAlienImage	Programs an entire Alien tag's memory image.	✓		
ProgAlienImageMap	The memory map type used when programming an entire Alien tag's memory image	✓		
ProgAlienImageNSI	The NSI word used when programming an entire Alien tag's memory image.	✓		

Program, Erase, and Verify Functions

The ProgramXXXX commands program a user-specified data into the tag's memory. The Erase command erases all data in the tag memory, setting all the tag EPC data to zero.

Programming functions always act on the single antenna specified by ProgAntenna. In multi-protocol readers, you specify a single tag protocol to use when programming with the ProgProtocol command.

ProgramEPC

9900 | 9680 | 9650

The ProgramEPC command allows the user to write a new EPC (Tag ID) into a tag. Once the command is issued, the reader verifies the presence of a tag, programs it, then reads back the tag memory to verify that the program command worked properly.

- Tags that have a locked EPC cannot be re-programmed. The tag's EPC bank must be Unlocked first.
- The ProgramEPC command expects you to supply an even number of hexadecimal bytes separated by spaces, representing the tag memory to program.
- Class1/Gen2 tags have a maximum EPC length, but smaller EPCs may still be programmed.
- The format of the response to a successful ProgramEPC command can be changed with the ProgSuccessFormat command.
- Programming can be performed using AutoMode by setting AutoAction=Program.
- In earlier reader firmware, this command was called simply "Program". In earlier reader models, this command was called "Program Tag". These command names still function for backward compatibility.
- If no argument is given in the ProgramEPC command, the reader instead uses the data specified by the ProgEPCData command (below).
- If the EPC to be programmed already matches the EPC stored in the tag, the ProgramEPC command simply returns a successful response, even if the tag is locked and *cannot* be programmed.

ProgramEPC Examples	
Condition	*Exactly one tag in field of view. Tag read is strong.* *Program a Class I tag with a new 64-bit ID.*
Command	`>ProgramEPC = 01 02 03 04 05 06 07 08 09 0A 0B 0C`
Response	`ProgramEPC = 01 02 03 04 05 06 07 08 09 0A 0B 0C`
Condition	*Tag on fringe of read range.*
Command	`> ProgramEPC = 01 02 03 04 05 06 07 08 09 0A 0B 0C`
Response	`Error 134: No tag found.`
Condition	*Tag is Locked*
Command	`> ProgramEPC = 01 02 03 04 05 06 07 08 09 0A 0B 0C`
Response	`Error 137: Tag Is locked.`

ProgramAndLockEPC

9900 | 9680 | 9650

The ProgramAndLockEPC command allows the user to write a new EPC code and lock the tag's EPC in one step. If the selected Class1/Gen2 lock type specified by ProgG2LockType is Lock (not PermaLock or PermaUnlock), then the Access password is also written to the tag. This is because simply locking the EPC bank doesn't protect that data unless there is a non-zero Access password in the tag. The reader uses the value given by ProgG2AccessPwd when writing the Access password. It is also recommended that you set the AcqG2AccessPwd to the same value, in order to facilitate this double operation.

- You provide the tag's new EPC code as an argument to the ProgramAndLockEPC command. If no data is given, the reader uses the ProgEPCData value instead.
- The ProgramAndLockEPC operation can be performed using AutoMode by setting AutoAction=ProgramAndLockEPC.
- If the EPC to be programmed already matches the EPC stored in the tag, the ProgramAndLockEPC command simply returns a successful response, even if the tag is locked and *cannot* be programmed.

ProgramAndLockEPC Examples

```
            // Set up
            >ProgG2LockType = Lock
            >ProgG2AccessPwd = F1 F2 F3 F4
            >AcqG2AccessPwd = F1 F2 F3 F4

            // Command with argument
Command     >ProgramAndLockEPC = 01 02 03 04 05 06 07 08 09 0A 0B 0C
Response    ProgramAndLockEPC = 0102 0304 0506 0708 090A 0B0C
            // With no argument
            >ProgEPCData = 01 02 03 04 05 06 07 08 09 0A 0B 0C

Command     >ProgramAndLockEPC
Response    ProgramAndLockEPC = 0102 0304 0506 0708 090A 0B0C
```

ProgramAccessPwd

9900 | 9680 | 9650

The ProgramAccessPwd command allows the user to write a new Access password to a Class1/Gen2 tag. A tag with a non-zero Access password will not allow memory banks to be locked or unlocked, or data in protected regions to be modified, unless the correct Access password is provided by the user (though the AcqG2AccessPwd command).

- You supply the tag's new Access password (four bytes) as an argument to the ProgramAccessPwd command. If no data is given, the reader uses the ProgG2AccessPwd value instead.
- Not all Class1/Gen2 tags may provide memory for an Access password. Those tags act as if they have an Access password of "0000 0000".
- In order to hide the Access password from others, you need to lock the AccessPwd field after writing the Access password to it.

ProgramAccessPwd Examples	
	// With an argument
Command	>ProgramAccessPwd = 01 02 03 04
Response	ProgramAccessPwd = 0102 0304
	// Without an argument
Command	>ProgG2AccessPwd = 01 02 03 04
Command	>ProgramAccessPwd
Response	ProgramAccessPwd = 0102 0304

ProgramKillPwd

9900 | 9680 | 9650

The ProgramKillPwd command allows the user to write a new Kill password to a Class1/Gen2 tag. Tags without a Kill password cannot be killed, and tags with a non-zero Kill password can only be killed if the stored password is provided in the Kill command.

- You provide the tag's new Kill password (four bytes) as an argument to the ProgramKillPwd command. If no data is given, the reader uses the ProgG2KillPwd value instead.
- Not all Class1/Gen2 tags support a Kill password. These tags may not be killed.
- In order to hide the Kill password from others, you need to lock the KillPwd field after writing the Kill password to it.

ProgramKillPwd Examples	
	// With an argument
Command	> ProgramKillPwd = FA FB FC FD
Response	ProgramKillPwd = FAFB FCFD
	// Without an argument
	>ProgG2KillPwd = FA FB FC FD
Command	> ProgramKillPwd
Response	ProgramKillPwd = FAFB FCFD

ProgramUser

9900 | 9680 | 9650

The ProgramUser command allows the user to program specified User data into a Class1/Gen2 tag.

- While it is possible to determine (bit 15 of the PC word) if User memory exists on a tag, this method only works if there is non-zero data in the first word of its User bank. There is no way of determining how much User memory a tag has, without attempting to access the User memory first.
- You provide the tag's new User memory as an argument to the ProgramUser command. If no data is provided, the reader uses the ProgUserData value instead.
- User memory must be written in word (2-byte) increments, so any supplied data must consist of an even number of bytes.

ProgramUser Examples	
Command	// With an argument >ProgramUser = DE AD BE EF CA FE C0 ED
Response	ProgramUser = DEAD BEEF CAFÉ C0ED
Command	// Without an argument >ProgUserData = F1 F2 F3 F4 F5 F6 F7 F8 > ProgramUser
Response	ProgramUser = F1F2 F3F4 F5F6 F7F8

Command	// Can read user data with generic "G2Read" command.
	// Bank=3, wordPtr=0, wordCnt=4
	>G2Read = 3, 0, 4

Response	G2Read = F1 F2 F3 F4 F5 F6 F7 F8

ProgramAndLockUser

9900 | 9680 | 9650

The ProgramAndLockUser command allows the user to write new User data and lock the tag's User bank in one step.

If the selected Class1/Gen2 lock type specified by ProgG2LockType is Lock (not PermaLock or PermaUnlock), then the Access password is also written to the tag. This is because simply locking the EPC bank doesn't protect that data unless there is a non-zero Access password in the tag. The reader uses the value given by ProgG2AccessPwd when writing the Access password. It is also recommended that you set the AcqG2AccessPwd to the same value, in order to facilitate this double operation.

- You provide the tag's new User data as an argument to the ProgramAndLockUser command. If no data is given, the reader uses the ProgUserData value instead.

ProgramAndLockUser Examples	
Command	// Command with argument >ProgramAndLockUser = F1 F2 F3 F4 F5 F6 F7 F8
Response	ProgramAndLockUser = F1F2 F3F4 F5F6 F7F8
Command	// With no argument >ProgUserData = F1 F2 F3 F4 F5 F6 F7 F8 > ProgramAndLockUser
Response	ProgramAndLockUser = F1F2 F3F4 F5F6 F7F8

G2Erase

9900 | 9680 | 9650

The G2Erase command erases (sets to 00) a specified portion of tag memory. You indicate which bank of memory, a word pointer for the first word to erase, and the number of words to erase. If the tag implements the Gen2 "BlockErase" command, it will write zeros to the indicated memory locations.

The format of the G2Erase command is:

```
G2Erase = <bank> <wordPtr> <wordLen>
```

- <bank> must be 0-3.
- <wordPtr> must be 0-2097151.
- <wordLen> must be 1-32 (issue successive commands to erase more).
- G2Erase is similar to Erase (see below), except Erase only writes zeros to the entire EPC, and applies to Gen1 tags as well..
- You cannot erase memory that has been locked, unless you have the correct Access password.

- If ProgDataUnit=Word, the G2Erase command is fundamentally the same as a G2Write of all zeros. When ProgDataUnit=Block, the Gen2 "BlockErase" command is issued instead, which is faster, but not supported by all tags.

G2Erase Example	
	// Erase both the Kill and Access passwords (four words in bank 0)
Command	>G2Erase = 0 0 4
Response	G2Erase = Success!
	// Compare with Program___Pwd, and G2Write
Command	>ProgramKillPwd = 00 00 00 00
Command	>ProgramAccessPwd = 00 00 00 00
Command	>G2Write = 0 0 00 00 00 00 00 00 00 00

Erase

9900 | 9680 | 9650

The Erase command attempts to erase the EPC memory of a tag in the reader's field of view. A tag affected by this command has its entire EPC tag memory set to zero. Once the command is issued, the reader verifies the presence of a tag, erases it, then reads back the tag memory to verify that the erase command worked properly.

- In Class1/Gen2, only one tag is erased, even if more than one tag is present.
- Tags that have a locked EPC bank cannot be erased.
- Don't confuse this command with the G2Erase command (above). G2Erase is a Gen2 command with more flexibility than Erase.
- Earlier readers used the command "Erase Tag", and current readers accept this for backward compatibility.

Erase Examples	
Command	>Erase
Response	Erase = Success!
	// Tag's EPC is locked
Command	>Erase
Response	Error 137: Tag is locked.

Lock, Unlock, and Kill Functions

Class1/Gen2 tags support three modes of password-based locking (Lock, PermaLock, PermaUnlock), on each of the tag's memory banks/fields (where supported): Kill Pwd, Access Pwd, EPC, User. A locked field may be unlocked, as long as the correct Access password is provided, and the field wasn't locked with a PermaLock or PermaUnlock lock type. Some tags may support PermaLocking individual blocks of User memory, and Alien Higgs3 & Higgs4 tags allow you to lock blocks of User memory in such a way that they cannot be read back without knowing the Access password. A Gen2 tag may be killed, provided it has a non-zero Kill Pwd, and the correct Kill password is provided. Killing a Gen2 tag renders it permanently inoperable.

LockEPC

9900 | 9680 | 9650

A tag's EPC can be locked to prevent its memory from being changed.

- The Class1/Gen2 LockEPC command takes a four-byte Access password, which must match the Access Pwd that already exists in the tag's memory. This Access password is not written to tag memory, but is required to overcome the tag's built-in security mechanism. In order to write the tag's Access Pwd field, use the ProgramAccessPwd command (below), or use ProgramAndLockEPC (above). If no argument is provided to the LockEPC command, the reader uses the value given by AcqG2AccessPwd instead.
- Once a Class 1/Gen 2 tag's EPC is locked, it may be unlocked, provided the original lock type wasn't PermaLock or PermaUnlock, and the correct Access password is provided.
- Class 1/Gen2 protocol allows for three mask types – Lock, PermaLock, and PermaUnlock. The type of locking that is performed is given by the ProgG2LockType command (below).
- Earlier reader firmware used the command "Lock Tag", which is still accepted for backward compatibility.
- Even earlier versions of reader firmware used the command "Lock" instead of "LockEPC", which is also still accepted.

C1G2 LockEPC Example	
Command Response	// With the current Access Pwd as an argument >LockEPC = 01 02 03 04 LockEPC = Success!
Command Response	// Without the current Access Pwd argument >AcqG2AccessPwd = 01 02 03 04 >LockEPC LockEPC = Success!

LockAccessPwd

9900 | 9680 | 9650

Locking the Access password of a Class1/Gen2 tag prevents the Access password from being changed (unless you provide the same password as the Access Key), or from being read back from the tag with a low-level read command.

- The Class1/Gen2 LockAccessPwd command takes a four-byte Access password, which must match the Access Pwd that already exists in the tag's memory. This Access password is not written to tag memory, but is required to overcome the tag's built-in security mechanism. If no argument is provided to the LockAccessPwd command, the reader uses the value given by AcqG2AccessPwd instead.
- In order to write the tag's Access Pwd field, use the ProgramAccessPwd command.
- Not all Class1/Gen2 tags implement the Access password.

LockAccessPwd Example	
Command Response	// With the current Access Pwd as an argument >LockAccessPwd = 01 02 03 04 LockAccessPwd = Success!
Command Response	// Without the current Access Pwd argument >AcqG2AccessPwd = 01 02 03 04 >LockAccessPwd LockEPC = Success!

LockKillPwd

9900 | 9680 | 9650

Locking the Kill password of a Class1/Gen2 tag prevents the Kill password from being changed (unless you provide the correct Access key), or from being read back from the tag with a low-level read command.

- The Class1/Gen2 LockKillPwd command takes a four-byte Access password, which must match the Access Pwd that already exists in the tag's memory. This Access password is not written to tag memory, but is required to overcome the tag's built-in security mechanism. If no argument is provided to the LockKillPwd command, the reader uses the value given by AcqG2AccessPwd instead.
- In order to write the tag's Kill Pwd field, use the ProgramKillPwd command.
- Not all Class1/Gen2 tags implement the Kill password.

LockKillPwd Example	
Command Response	// With the current Access Pwd as an argument > LockKillPwd = 01 02 03 04 LockKillPwd = Success!
Command Response	// Without the current Access Pwd argument >AcqG2AccessPwd = 01 02 03 04 > LockKillPwd LockKillPwd = Success!

LockUser

9900 | 9680 | 9650

Locking the User bank of a Class1/Gen2 tag prevents the User memory from being changed (unless you provide the same password as the Access Key). Locking the User bank does not prevent other users from reading the User data.

- The Class1/Gen2 LockUser command takes a four-byte Access password, which must match the corresponding Access Pwd that already exists in the tag's memory. This Access password is not written to tag memory, but is required to overcome the tag's built-in security mechanism. If no argument is provided to the LockUser command, the reader uses the value given by AcqG2AccessPwd instead.
- In order to write the tag's User data, use the ProgramUser or ProgramAndLockUser commands.
- Not all Class1/Gen2 tags implement User memory.

LockUser Example	
Command Response	// With the current Access Pwd as an argument > LockUser = 01 02 03 04 LockUser = Success!
Command Response	// Without the current Access Pwd argument >AcqG2AccessPwd = 01 02 03 04 > LockUser LockUser = Success!

LockUserBlocks

9900 | 9680 | 9650

One optional feature in the Gen2 protocol gives you the ability to **permalock** individual blocks of User memory (called BlockPermaLock). Alien Higgs3 and Higgs4 tags support this feature, and you set these permalocks with the LockUserBlocks reader command.

Please read the Gen2 protocol specification for details on this feature, and the ramifications of permalocking blocks of User memory. Once something is permalocked, it can never be changed or unlocked again, and the interaction between block permalocks and User bank locks can be complicated.

The Gen2 protocol allows tag vendors to subdivide the User memory bank into blocks (the same blocks referred to in the HideAlienUserBlocks command, below), and how much data goes into each block is determined by the vendor. Alien Higgs3 and Higgs4 tags have blocks four words (8 bytes, or 64 bits) in size, so the maximum 512-bits of available User memory is subdivided into eight blocks.

The LockUserBlocks command takes three arguments – a decimal block pointer, followed by two hex bytes that are bitmask representations of the desired permalock states of each block of User memory. A "1" bit indicates that you wish to permalock the associated User block. A "0" bit means leave the current permalock state (locked or unlocked) alone.

```
LockUserBlocks = <blockPtr> <blockmask1> <blockmask2>
```

The underlying Gen2 command allows the reader to configure User block permalocks in groups of sixteen blocks at a time, so the LockUserBlocks command requires two complete <blockmask> bytes (16 bits total). The <blockPtr> allows you to address higher groups of 16 blocks, and should therefore be a multiple of 16 (0, 16, 32, etc.).

Alien Higgs3 tags implement only eight blocks of User data (Higgs4 have four blocks), so the <blockPtr> should always be zero and the <blockmask2> value should also be zero. Attempting to permalock User blocks that don't exist results in a tag "memory overrun" error and the entire operation will fail.

- Not all Class1/Gen2 tags implement User memory, or the BlockPermaLock Gen2 feature.
- You may issue the LockUserBlocks command repeatedly on the same set of User blocks – locking individual blocks each time.
- Block permalocks are irreversible – use with caution!

LockUserBlocks Example	
	// Permalock the first two User blocks (C0 = 11000000)
Command	>LockUserBlocks = 0 C0 00
Response	LockUserBlocks = Success!
	// Write 3 blocks
Command	>ProgramUser = A1 A2 A3 A4 A5 A6 A7 A8 B1 B2 B3 B4 B5 B6 B7 B8 C1 C2 C3 C4 C5 C6 C7 C8
Response	Error 137: Tag is locked.
	// Still can write to block #3 (word #8)
Command	>G2Write = 3 8 C1 C2 C3 C4 C5 C6 C7 C8
Response	G2Write = Success!
Command	>G2Read = 3 0 0
Response	G2Read = 11 12 13 14 15 16 17 18 21 22 23 24 25 26 27 28 C1 C2 C3 C4 C5 C6 C7 C8
	// Now additionally lock the 3rd User block (20 = 00100000)
Command	>LockUserBlocks = 0 20 00
Response	LockUserBlocks = Success!
	// The first three User blocks are now permalocked.

HideAlienUserBlocks

9900 | 9680 | 9650

Alien Higgs3 and Higgs4 tags implement a feature which allows you to lock individual blocks of User memory so that they cannot be read back without knowing the Access password. These "read locks" allow you to store sensitive data in the tag, which only you and trusted partners can read or modify.

Alien Higgs3 tags subdivide User memory into 4-word blocks (the same blocks referred to in the LockUserBlocks command, above). The maximum amount of User memory a Higgs3 tag can have is 512 bits, which corresponds to eight blocks. Alien Higgs4 tags subdivide User memory into 2-word blocks. The maximum amount of User memory a Higgs4 tag can have is 128 bits which corresponds to four blocks.

HideAlienUserBlocks takes a single hex byte as an argument, and it is a bitmask representing the desired read locks on all eight User blocks. The most-significant bit in the bitmask represents the first User block, and so on, which means that to place a read lock on block #1, the bitmask would be 80 (10000000_{binary}), not 01 (00000001_{binary}).

- Read locks are only enforced if the tag has a non-zero Access password stored in it.
- Only Alien Higgs3 and Higgs4 tags implement this feature.

HideAlienUserBlocks Example

```
                // Read 1st three blocks (12 words) of User data
Command         >G2Read = 3 0 12
Response        G2Read = 11 12 13 14 15 16 17 18 21 22 23 24 25 26 27 28 C1 C2 C3
                C4 C5 C6 C7 C8

                // Read-lock the 1st and 3rd blocks (A0 = 10100000)
Command         >HideAlienUserBlocks = A0
Response        HideAlienUserBlocks = Success!

                // No Access Pwd yet, so it is still readable:
Command         >G2Read = 3 0 12
Response        G2Read = 11 12 13 14 15 16 17 18 21 22 23 24 25 26 27 28 C1 C2 C3
                C4 C5 C6 C7 C8

                // Write non-zero Access Pwd and read again
Command         >ProgramAccessPwd = 11 22 33 44
Response        ProgramAccessPwd = 1122 3344

Command         >G2Read = 3 0 12
Response        Error 260: (Tag Error) Memory locked.

                // Can read block #2 (starting at word #4)
Command         >G2Read = 3 4 4
Response        G2Read = 21 22 23 24 25 26 27 28

                // …but not blocks #1,3
Command         >G2Read = 3 0 4
Response        Error 260: (Tag Error) Memory locked.
Command         >G2Read = 3 8 4
Response        Error 260: (Tag Error) Memory locked.

                // Supplying the correct Access Pwd lets us see everything
Command         >AcqG2AccessPwd = 11 22 33 44
Response        AcqG2AccessPwd = 11 22 33 44

                >G2Read = 3 0 12
Command         G2Read = 11 12 13 14 15 16 17 18 21 22 23 24 25 26 27 28 C1 C2 C3
Response        C4 C5 C6 C7 C8
                // Reveal the hidden blocks
Command         >HideAlienUserBlocks = 00
Response        HideAlienUserBlocks = Success!

                // Don't need the password now
Command         >AcqG2AccessPwd = 0 0 0 0
Response        AcqG2AccessPwd = 00 00 00 00

Command         >G2Read = 3 0 12
Response        G2Read = 11 12 13 14 15 16 17 18 21 22 23 24 25 26 27 28 C1 C2 C3
                C4 C5 C6 C7 C8
```

UnlockEPC

9900 | 9680 | 9650

The UnlockEPC command unlocks a Class1/Gen2 tag's EPC memory bank.

- The UnlockEPC command takes a four-byte Access password, which must match the Access Pwd that already exists in the tag's memory. This Access password is not written to tag memory, but is required to overcome the tag's built-in security mechanism. If no argument is provided to the UnlockEPC command, the reader uses the value given by AcqG2AccessPwd instead.
- Class 1/Gen 2 tags whose EPC bank has been locked with the PermaLock lock type may not be unlocked.

UnlockEPC Example	
	// With the current Access Pwd as an argument
Command	>UnlockEPC = 01 02 03 04
Response	UnlockEPC = Success!
	// Without the current Access Pwd argument
	>AcqG2AccessPwd = 01 02 03 04
Command	>UnlockEPC
Response	UnlockEPC = Success!

UnlockAccessPwd

9900 | 9680 | 9650

The UnlockAccessPwd command unlocks a Class1/Gen2 tag's Access Pwd field. This command does not apply to Class1/Gen1 tags.

- The UnlockAccessPwd command takes a four-byte Access password, which must match the corresponding Access Pwd that already exists in the tag's memory. This Access password is not written to tag memory, but is required to overcome the tag's built-in security mechanism. If no argument is provided to the UnlockAccessPwd command, the reader uses the value given by AcqG2AccessPwd instead.
- Class 1/Gen 2 tags whose Access Pwd has been locked with the PermaLock lock type may not be unlocked.
- The Access password is not a mandatory Class1/Gen2 feature - some tags may not implement it.

UnlockAccessPwd Example	
	// With the current Access Pwd as an argument
Command	> UnlockAccessPwd = 01 02 03 04
Response	UnlockAccessPwd = Success!
	// Without the current Access Pwd argument
	>AcqG2AccessPwd = 01 02 03 04
Command	> UnlockAccessPwd
Response	UnlockAccessPwd = Success!

UnlockKillPwd

9900 | 9680 | 9650

The UnlockKillPwd command unlocks a Class1/Gen2 tag's Kill Pwd field. This command does not apply to Class1/Gen1 tags.

- The UnlockKillPwd command takes a four-byte Access password, which must match the corresponding Access Pwd that already exists in the tag's memory. This Access password is not written to tag memory, but is required to overcome the tag's built-in security mechanism. If no argument is provided to the UnlockKillPwd command, the reader uses the value given by AcqG2AccessPwd instead.
- Class 1/Gen 2 tags whose Kill Pwd has been locked with the PermaLock lock type may not be unlocked.
- The Kill password is not a mandatory Class1/Gen2 feature - some tags may not implement it.

UnlockKillPwd Example	
	// With the current Access Pwd as an argument
Command	> UnlockKillPwd = 01 02 03 04
Response	UnlockKillPwd = Success!
	// Without the current Access Pwd argument
	>AcqG2AccessPwd = 01 02 03 04
Command	> UnlockKillPwd
Response	UnlockKillPwd = Success!

UnlockUser

9900 | 9680 | 9650

The UnlockUser command unlocks the User bank of a Class1/Gen2 tag. This command does not apply to Class1/Gen1 tags.

- The UnlockUser command takes a four-byte Access password, which must match the corresponding Access Pwd that already exists in the tag's memory. This Access password is not written to tag memory, but is required to overcome the tag's built-in security mechanism. If no argument is provided to the UnlockUser command, the reader uses the value given by AcqG2AccessPwd instead.
- Class 1/Gen 2 tags whose User bank has been locked with the PermaLock lock type may not be unlocked.
- User memory is not a mandatory Class1/Gen2 feature - some tags may not implement it.

UnlockUser Example	
	// With the current Access Pwd as an argument
Command	> UnlockUser = 01 02 03 04
Response	UnlockUser = Success!
	// Without the current Access Pwd argument
	>AcqG2AccessPwd = 01 02 03 04
Command	> UnlockUser
Response	UnlockUser = Success!

HideAlienUserBlocks

9900 | 9680 | 9650

Alien Higgs3 tags implement a feature which allows you to lock individual blocks of User memory so that they cannot be read back without knowing the Access password. These "read locks" allow you to store sensitive data in the tag, which only you and trusted partners can read or modify.

Alien Higgs3 tags subdivide User memory into 4-word blocks (the same blocks referred to in the LockUserBlocks command, above). The maximum amount of User memory a Higgs3 tag can have is 512 bits, which corresponds to eight blocks.

HideAlienUserBlocks takes a single hex byte as an argument, and it is a bitmask representing the desired read locks on all User blocks. The most-significant bit in the bitmask represents the first User block, and so on, which means that to place a read lock on block #1, the bitmask would be 80 (10000000), not 01 (00000001).

- Read locks are only enforced if the tag has a non-zero Access password stored in it.

- Only Alien Higgs3 and Higgs4 tags implement this feature.

HideAlienUserBlocks Example

```
                // Read 1st three blocks (12 words) of User data
Command         >G2Read = 3 0 12
Response        G2Read = 11 12 13 14 15 16 17 18 21 22 23 24 25 26 27 28 C1 C2 C3
                C4 C5 C6 C7 C8

                // Read-lock the 1st and 3rd blocks (A0 = 10100000)
Command         >HideAlienUserBlocks = A0
Response        HideAlienUserBlocks = Success!

                // No Access Pwd yet, so it is still readable:
Command         >G2Read = 3 0 12
Response        G2Read = 11 12 13 14 15 16 17 18 21 22 23 24 25 26 27 28 C1 C2 C3
                C4 C5 C6 C7 C8

                // Write non-zero Access Pwd and read again
Command         >ProgramAccessPwd = 11 22 33 44
Response        ProgramAccessPwd = 1122 3344

Command         >G2Read = 3 0 12
Response        Error 260: (Tag Error) Memory locked.

                // Can read block #2, but not blocks #1,3
Command         >G2Read = 3 4 4
Response        G2Read = 21 22 23 24 25 26 27 28

Command         >G2Read = 3 0 4
Response        Error 260: (Tag Error) Memory locked.

Command         >G2Read = 3 8 4
Response        Error 260: (Tag Error) Memory locked.

                // Supplying the correct Access Pwd lets us see it
Command         >AcqG2AccessPwd = 11 22 33 44
Response        AcqG2AccessPwd = 11 22 33 44

Command         >G2Read = 3 0 12
Response        G2Read = 11 12 13 14 15 16 17 18 21 22 23 24 25 26 27 28 C1 C2 C3
                C4 C5 C6 C7 C8
```

Kill

9900 | 9680 | 9650

When applied to a Class1/Gen2 tag, the tag is permanently disabled – it won't respond to any subsequent reader commands. Class1/Gen2 tags can only be killed if they implement the Kill Pwd, the current Kill Pwd is non-zero, and the user includes the correct Kill Pwd in the Kill command.

Earlier readers used the command, "Kill Tag", which newer readers also accept for backward compatibility.

- You can specify the full tag EPC followed by the four-byte Kill Password that was previously written to the tag with the ProgramKillPwd command. Alternatively, if you leave off the EPC and just provide the four-byte Kill Password, the reader attempts to kill the first Class1/Gen2 tag it finds. If you use the Kill command without arguments, the reader attempts to kill the first Class1/Gen2 tag it finds, using the Kill password you specify with the ProgG2KillPwd command.
- Access Key is not required, just the Kill Pwd.
- Tag must have a non-zero Kill Pwd written already.
- Tag is rendered permanently inoperative.

C1G2 Kill Example	
Condition	*EPC is B0 B0 11 11 22 22 33 33 44 44 55 55* *Programmed Kill password is 0A 0B 0C 0D*
	// Specify EPC to kill and its kill password
Command	>Kill = B0 B0 11 11 22 22 33 33 44 44 55 55 0A 0B 0C 0D
Response	Kill = Success!
	// Specify just the kill password (kills first tag it finds)
Command	>Kill = 0A 0B 0C 0D
Response	Kill = Success!
	// Preload kill password to use, then Kill without arguments
Command	>ProgG2KillPwd = 0A 0B 0C 0D
Response	ProgG2KillPwd = 0A 0B 0C 0D
Command	>Kill
Response	Kill = Success!

Programming Configuration and Data Storage Commands

The following "ProgXXXX" commands allow you to configure key parameters which govern all programming operations, and preset certain data fields to allow for "argument-less" programming operations, including automatic incrementing of EPC or User data. These commands allow programming operations to occur in AutoMode, when the reader must have all of the required data up-front in order to be able to operate independently.

The two most important of these commands are the ProgProtocol and ProgAntenna commands.

ProgProtocol

9900 | 9680 | 9650

For readers that support programming in more than one air protocol, the reader must be told ahead of time which protocol to use in programming operations. You do this with the ProgProtocol command. In contrast to the TagType command (which is a bit mask of protocols to use during a tag read), the ProgProtocol takes a value indicating the single protocol to use.

The following values have been used at some point, but current Alien readers only support the Class 1/Gen 2 standard:

0 = Class 0
1 = Class 1/Gen 1
2 = Class 1/Gen 2 (default)

- It is recommended when you change the ProgProtocol, that you also change the TagType setting to includes the selected ProgProtocol. This way after you perform a programming operation, if you subsequently use a "get TagList" command to verify, you are reading with the same protocol used to program.

ProgProtocol Examples	
Command	>ProgProtocol?
Response	ProgProtocol = 2
Command	>ProgProtocol = 1
Response	ProgProtocol = 1

ProgAntenna

9900 | 9680 | 9650

Programming tags requires a controlled, repeatable RF environment (see previous section titled, "Programming Distance and Power Levels"), and in most cases, programming commands can potentially affect all the tags in the RF field. For this reason, rather than using the AntennaSequence to indicate which antenna(s) to issue programming commands on, you use the ProgAntenna attribute instead.

All programming commands are issued on the antenna port specified by ProgAntenna, regardless of the current AntennaSequence value.

- Allowed Values: Integer(0..3) (max antenna depends on reader model)
- Default Value: 0
- Only one antenna value is allowed for ProgAntenna, as opposed to the AntennaSequence, which can take a list of antenna numbers.
- It is recommended when setting the ProgAntenna, that you also set an AntennaSequence value that includes the selected ProgAntenna. This way after you perform a programming operation, if you subsequently use a "get TagList" command to verify, you are reading on the same antenna used to program.

ProgAntenna Examples	
Command Response	>ProgAntenna? ProgAntenna = 0
Command Response	>ProgAntenna = 2 ProgAntenna = 2

ProgG2NSI

9900 | 9680 | 9650

The ProgG2NSI parameter allows programming custom NSI values of the tag's PC word when using the ProgramEPC command. The parameter value consists of two bytes in hex format to combine into 9 bits of NSI. An NSI value greater than 0x01FF indicates that there is no need to change the NSI part of the PC word. The format is as follows:

```
ProgG2NSI = <hexbyte1> <hexbyte2>
```

where `<hexbyte1,2>` is between 0x00 and 0xFF. If the `<hexbyte1>` value is greater than 1 (i.e. the NSI value is greater than 0x01FF) then the NSI part of the PC word will not be changed during programming. The default value is '`FF FF`'

- Value supplied must be 2 bytes in hexadecimal format.
- The default value of ProgG2NSI is '`FF FF`'.
- If the value of the first byte is greater than 1 then the NSI part of the PC word will not be changed during programming.

ProgG2NSI Examples	
Command Response	>ProgG2NSI? ProgG2NSI = FF FF

Command
Response
	>ProgG2NSI = 01 23
	ProgG2NSI = 01 23
	
	// look at existing NSI value (bank #1, word #1)
	>G2Read = 1 1 1
	G2Read = 30 00
	// NSI is all zeros
	
	// program the tag
	>ProgramEPC
	ProgramEPC = 0102 0304 0506 0708 090A 0B0C
	
	// verify that the NSI part of the PC word has changed
	>G2Read = 1 1 1
	G2Read = 31 23
	// NSI is set to 0x123

ProgEPCData

9900 | 9680 | 9650

The ProgEPCData value is used when the ProgramEPC and ProgramAndLockEPC commands are used without arguments, as well as in AutoMode when a program operation has been selected as the AutoAction, such as "ProgramEPC" or "ProgramAndLockEPC". This command allows you to specify the next EPC code to program into a tag.

- Value supplied should be an even number of bytes for Class1/Gen2 (max EPC length depends on tag implementation).
- The default ProgEPCData is 00 00 00 00 00 00 00 00 00 00 00 00.
- The value may be optionally incremented (in a hexadecimal fashion) following the operation, depending on the ProgEPCDataInc setting.
- Earlier readers used the command "ProgramID" instead, which is still accepted for backward compatibility.
- The special case of "ProgEPCData = None" causes the reader to "clear" the tag's EPC when it programs. This sets the EPC length to zero, so that the tag has no EPC (the factory state for EPC tags, according to the Gen2 specification). The tag still responds to inventories, but the TagList entry will not show an EPC code.

ProgEPCData Examples	
Command Response	>ProgEPCData = 01 23 45 67 89 AB CD EF ProgEPCData = 01 23 45 67 89 AB CD EF
Command Response	>ProgEPCData? ProgEPCData = 01 23 45 67 89 AB CD EF

ProgUserData

9900 | 9680 | 9650

The ProgUserData value is used when the ProgramUser command is used without an argument. It is simply a shortcut placeholder for data which allows you to write the same User data to multiple tags without having to enter the same data each time.

- Value supplied must be an even number of bytes. The maximum length is determined by the tag implementation.
- The default ProgUserData is 00 00.
- The value may be optionally incremented (in a hexadecimal fashion) following a ProgramUser operation, depending on the ProgUserDataInc value.

ProgUserData Examples	
Command	>ProgUserData?
Response	ProgUserData = 00 00
Command	>ProgUserData = 01 23 45 67 89 AB CD EF
Response	ProgUserData = 01 23 45 67 89 AB CD EF

ProgEPCDataInc

9900 | 9680 | 9650

The ProgEPCDataInc command allows you to specify under which conditions the ProgEPCData value is incremented, either following a ProgramEPC command, or when using AutoMode with a program operation selected as the AutoAction, such as "ProgramEPC".

Value	Description
OFF	ProgEPCData is never incremented.
Success	ProgEPCData is incremented only when ProgramEPC succeeds.
Fail	ProgEPCData is incremented only when ProgramEPC fails.
Always	ProgEPCData is incremented after each ProgramEPC command.
Write	ProgEPCData is incremented only when a write operation is attempted. This reduces the number of 'false' increments because the data value is not incremented in cases when there are no tags in the field or a tag was not singulated. **This option is not supported on the ALR-9680 & 9650.**

This command provides more functionality than the older, deprecated ProgIncrementOnFail command, but ProgIncrementOnFail has been retained for backward compatibility. The two commands are coupled, in that if you set a ProgEPCDataInc value that causes failures to increment ("Fail" or "Always"), then ProgIncrementOnFail is set to ON, and if you set a ProgEPCDataInc value that causes failures to not increment ("OFF" or "Success"), then ProgIncrementOnFail is set to OFF. The reverse is also true – changing the ProgIncrementOnFail may have a corresponding effect on the ProgEPCDataInc value.

- The default ProgEPCDataInc value is Success.
- ProgEPCDataInc is incremented in a hexadecimal fashion.

ProgEPCDataInc Examples	
Command	>ProgEPCDataInc?
Response	ProgEPCDataInc = Success
Command	>ProgEPCDataInc = Always
Response	ProgEPCDataInc = Always

ProgUserDataInc

9900 | 9680 | 9650

The ProgUserDataInc command allows you to specify under which conditions the ProgUserData value is incremented following the ProgramUser command.

Value	Description
OFF	ProgUserData is never incremented.
Success	ProgUserData is incremented only when ProgramUser succeeds.
Fail	ProgUserData is incremented only when ProgramUser fails.
Always	ProgUserData is incremented after each ProgramUser command.
Write	ProgUserData is incremented only when a write operation is attempted. This reduces the number of 'false' increments because the data value is not incremented in cases when there are no tags in the field or a tag was not singulated. **This option is not supported on the ALR-9680 & 9650.**

- The default value is Success.
- ProgUserData is incremented in a hexadecimal fashion.

ProgUserDataInc Examples	
Command	>ProgUserDataInc?
Response	ProgUserDataInc = Success
Command	>ProgUserDataInc = OFF
Response	ProgUserDataInc = OFF

ProgEPCDataIncCount

9900 | 9680 | 9650

The `ProgEPCDataIncCount` command limits the number of `ProgEPCData` increments:

```
ProgEPCDataIncCount = <count>
```

where the value of `<count>` is as follows:

`-1 (default)` : keep incrementing with no limit.
`0` : stop incrementing and do not allow programming until the counter is reset to `-1` or to a positive value.
`1 to 86400000`: number of increments left.

The counter cannot be saved with the '`Save`' command, it is always initialized to `-1` on reader startup.

The current value of the counter indicates the number of increments left. Each time `ProgEPCData` is incremented, the `ProgEPCDataInc` is reduced by one, until it reaches zero.

When you program in manual mode (using the `ProgramEPC` command) and the counter hits `0`, any attempt to program a tag will result in an error unless `ProgEPCDataInc` is set to `OFF`. When you program in `AutoMode` and the counter hits `0`, a notification is sent (`NotifyMode` must be turned `ON`) that the counter has reached 0 and then `AutoMode` gets turned `OFF`.

Note that `AutoMode` turns `OFF` without an explicit user command. This is similar to the behavior of the `NotifyRetryCount` when sending notification will cease after the `NotifyRetryCount` number of retries.

If AutoMode is turned ON without resetting the counter it will perform one AutoAction cycle and then will turn itself OFF.

This behavior only applies to the AutoAction = ProgramXXXX options.

ProgEPCDataIncCount Examples	
Command	> ProgEPCDataIncCount?
Response	ProgEPCDataIncCount = -1
Command	> ProgEPCDataIncCount = 100
Response	ProgEPCDataIncCount = 100

ProgUserDataIncCount

9900 | 9680 | 9650

The ProgUserDataIncCount command limits the number of ProgUserData increments.

```
ProgUserDataIncCount = <count>
```

where the value of <count> is as follows:

-1 (default) : keep incrementing with no limit.
0 : stop incrementing and do not allow programming until the counter is reset to -1 or to a positive value.
1 to 86400000 : number of increments left.

The counter cannot be saved with the 'Save' command, it is always initialized to -1 on reader startup.

The current value of the counter indicates the number of increments left. Each time ProgUserData is incremented, the ProgUserDataInc is reduced by one, until it reaches zero.

When you program in manual mode (using the ProgramUser command) and the counter hits 0, any attempt to program a tag will result in an error unless ProgUserDataInc is set to OFF. When you program in AutoMode and the counter hits 0, a notification is sent (NotifyMode must be turned ON) that the counter has reached 0 and then AutoMode gets turned OFF.

Note that AutoMode turns OFF without an explicit user command. This is similar to the behavior of the NotifyRetryCount when sending notification will cease after the NotifyRetryCount number of retries.

If AutoMode is turned ON without resetting the counter it will perform one AutoAction cycle and then will turn itself OFF.

This behavior only applies to the AutoAction = ProgramUser options.

ProgUserDataIncCount Examples	
Command	> ProgUserDataIncCount?
Response	ProgUserDataIncCount = -1
Command	> ProgUserDataIncCount = 100
Response	ProgUserDataIncCount = 100

ProgG2AccessPwd

9900 | 9680 | 9650

The ProgG2AccessPwd value is used when the ProgramAccessPwd command is used without an argument, as well as in AutoMode when an AutoAction of ProgramAndLockEPC is used (when ProgG2LockType=Lock, the Access password is automatically written when the tag is locked).

This command is different from the similar-sounding AcqG2AccessPwd command. ProgG2AccessPwd is a placeholder for an Access Pwd value to be written to the tag, whereas the AcqG2AccessPwd provides the tag's current Access Pwd to be used when operating on a Class 1/Gen2 tag that is already password-protected.

- Value supplied must be a 4-byte value.
- The default ProgG2AccessPwd is 00 00 00 00.

ProgG2AccessPwd Examples	
Command	`>ProgG2AccessPwd = 1 2 3 4`
Response	`ProgG2AccessPwd = 01 02 03 04`
Command	`>ProgG2AccessPwd?`
Response	`ProgG2AccessPwd = 01 02 03 04`

ProgG2KillPwd

9900 | 9680 | 9650

The ProgG2KillPwd value is used when the ProgramKillPwd command is used without an argument. It is simply a shortcut placeholder for data which allows you to write the same Kill password to multiple tags without having to enter the same password each time.

- Value supplied must be a 4-byte value.
- The default ProgG2KillPwd is 00 00 00 00.

ProgG2KillPwd Examples	
Command	`>ProgG2KillPwd = F1 F2 F3 F4`
Response	`ProgG2KillPwd = F1 F2 F3 F4`
Command	`>ProgG2KillPwd?`
Response	`ProgG2KillPwd = F1 F2 F3 F4`

ProgG2LockType

9900 | 9680 | 9650

The Class 1/Gen 2 protocol allows each of the tag's memory banks/fields to be locked with one of three lock types. The ProgG2LockType command allows you to specify which lock type to use in the next Lock operation.

Lock Type	Description
Lock	The data is locked, and may be subsequently unlocked.
PermaLock	The data is locked, and may never be changed or unlocked.
PermaUnlock	The data is unlocked, and may never be locked.

- The default ProgG2LockType is Lock.

- Technically, there is a fourth lock type defined by the Class 1/Gen 2 protocol – "Unlocked". Rather than "lock with the Unlocked lock type", Alien readers provide explicit Unlock___ commands.
- The PermaLock and PermaUnlock types are, well, permanent.
- Locking the EPC or User memory banks do not prevent them from being read. Locking the Kill password or Access password does prevent those fields from being read.
- Locking the EPC or User memory banks does not prevent them from being changed, unless there is a non-zero Access password stored in the tag.
- Even if there is a non-zero Access password stored in the tag, locked banks may still be changed if the correct Access key is provided (with the AcqG2AccessPwd command). The exception is permalocked data, which can't be changed even with the correct Access key.

ProgG2LockType Examples	
Command	`>ProgG2LockType?`
Response	`ProgG2LockType = Lock`
Command	`>ProgG2LockType = PermaLock`
Response	`ProgG2LockType = PermaLock`

ProgDataUnit

9900 | 9680 | 9650

The usual way of writing data to a Gen2 tag is to write a word (16 bits) of data, then write the next word, then the next, etc., perhaps verifying each word by reading it back. An alternative way to write to Gen2 tags is a "block write", where the reader hands the tag a list of words to be written sequentially into memory. A block write is significantly faster than writing word-by-word.

The Alien reader takes care of the details required to issue both types of writes, you just have to tell it which mode to use, using the ProgDataUnit command. All of the existing programming commands, ProgramEPC, ProgramUser, G2Write, and even programming in AutoMode, automatically switch to the block-write method if you specify this.

Since the BlockWrite command is an optional Gen2 feature, not all tags support it. Alien Higgs3 and Higgs4 tags are the first Alien tags to implement this feature. **If you attempt to perform a block write to a tag that doesn't implement BlockWrite, the operation will fail. You should use BlockWrite only if you know the tag supports it.**

- Allowed Values: "Word" | "Block"
- Default Value: "Word"

ProgDataUnit Examples	
Command	`>ProgDataUnit?`
Response	`ProgDataUnit = Word`
Command	`>ProgDataUnit = Block`
Response	`ProgDataUnit = Block`

ProgBlockSize

9900 | 9680 | 9650

When programming in block mode (`ProgDataUnit = block`), the reader can write data to the tag more quickly by sending more data with each transaction. The `ProgBlockSize` command lets you specify the

size of the block. Alien Higgs tags allow you to write up to 32 words at a time, whereas other tag manufacturers may allow only 2 words.

Trying to write data that extends across block boundaries will result in a failure of the write operation (see the ProgBlockAlign command, following this one).

The default value for `ProgBlockSize` is 0 (i.e. “default”), which corresponds to 32 words.

- Allowed Values: 0, 2, 4, 8, 16, 32
- Default Value: 0 (reader automatically picks a reasonable value, currently 32)

ProgBlockSize Examples	
Command	`>ProgBlockSize?`
Response	`ProgBlockSize = 0`
Command	`>ProgBlockSize = 2`
Response	`ProgBlockSize = 2`

ProgBlockAlign

9900 | 9680 | 9650

When programming in Block mode, you usually have to take care to not write across block boundaries (block sizes vary from tag manufacturer to tag manufacturer), or the tag will respond with an error. The `ProgBlockAlign=On/Off` command, when turned on, will tell the reader to automatically break up the data you wish to write to the tag so that each individual piece can be block-written to the tag separately, thus avoiding the errors associated with writing across block boundaries.

Note that for Alien Higgs tags, the block size is 32 words, which is never smaller than the size of the largest memory bank (USER), so you will never encounter block-boundary issues, unless you use a tag from a different manufacturer.

- Allowed Values: On | Off
- Default Value: Off

ProgBlockAlign Examples	
Command	`>ProgBlockAlign?`
Response	`ProgBlockAlign = Off`
Command	`>ProgBlockAlign = On`
Response	`ProgBlockAlign = On`

ProgAttempts

9900 | 9680 | 9650

The ProgAttempts attribute specifies the number of attempts the reader should make to program a tag during a Program Tag operation.

- Allowed Values: Integer(1..7)
- Default Value: 3

ProgAttempts Examples	
Command Response	>ProgAttempts? ProgAttempts = 3
Command Response	>ProgAttempts = 5 ProgAttempts = 5

ProgSuccessFormat

9900 | 9680 | 9650

The "Program Tag" command returns the ID that was programmed into the tag if the programming attempt was successful. Older versions of the firmware simply returned "Success!". You can select which behavior the reader exhibits with the ProgSuccessFormat value.

0 = "Success!"
1 = the programmed tag ID (default value)

ProgSuccessFormat Examples	
Command Response	>ProgSuccessFormat? ProgSuccessFormat = 0 >Program Tag = 11 22 33 44 55 66 77 88 Program Tag = Success!
Command Response	>ProgSuccessFormat = 1 ProgSuccessFormat = 1 >Program Tag = 11 22 33 44 55 66 77 88 Program Tag = 11 22 33 44 55 66 77 88

ProgSingulate

9900 | 9680 | 9650

When a programming operation is attempted and there is more than one tag in the field the result is unpredictable: the programming may succeed or may fail due to multiple tags responding at the same time.
When the ProgSingulate option is turned ON then every time a programming operation is attempted a short inventory is performed right before programming and if there are 2 or more tags in the field then the "Singulation Error 149" is returned without an attempt to program.

The format is as follows:

ProgSingulate = ON | OFF

The command is applicable to all Gen2 programming commands (including AutoMode AutoActions) except for the G2Write and G2Erase. The default value is OFF.

- Allowed Values: "ON" | "OFF"
- Default Value: "OFF"

ProgSingulate Examples	
Command Response	>ProgSingulate? ProgSingulate = OFF
Command Response	// set to check-if-singulated before an attempt to program >ProgSingulate = ON ProgSingulate = ON >ProgramEPC Error 149: Singulation error > t Tag:E200 3411 B801 0108 6500 6464, ... Tag:E200 3411 B801 0108 6500 6465, ...

Low-Level Programming – TagInfo, G2Read, G2Write

Class 1/Gen 2 tags contain many memory fields, and you can conveniently write to these fields using the commands described previously – ProgramEPC, ProgramAccessPwd, ProgramKillPwd, and ProgramUser – but you may find the need to arbitrarily write to specific portions of tag memory. The low-level G2Write command can be used to do this.

Also, the reader only exposes the tag's EPC field (and, through a custom TagList token, the PC word) in its TagList data. In order to query a tag for its Access or Kill passwords, User data, NSI field, etc. you must use the low-level G2Read command.

The G2Read and G2Write commands both operate on word-boundaries in the tag memory – you cannot read or write individual bits or bytes of data. They also require you to specify the bank and word pointers into tag memory, so you must have an understanding of the structure of tag memory in order to use them. The bank numbers and word pointers into each bank all begin at zero.

Recall:

G2 bank 0 - RESERVED (Kill and Access passwords)
G2 bank 1 - EPC (CRC + PC + EPC)
G2 bank 2 - TID (Tag identifier)
G2 bank 3 - USER (user-specified data, not supported by all tags)

The TagInfo reads the TID bank and some other parts of the tag, and returns to you a summary of the lock states of the various fields, manufacturer and tag model, PC word, availability of User memory, etc.

These commands operate only on a tag that matches the current acquire mask, and they allow access into any bank of tag memory that the tag supports. The reader performs these commands on the current ProgAntenna.

If the memory bank you are trying to access is locked, you will need to supply the correct access password (with AcqG2AccessPwd).

TagInfo

9900 | 9680 | 9650

The TagInfo command gives you information about a tag, such as the manufacturer ID, tag model & version. For Alien tags, the TagInfo is extended, to include information such as lock states of each of its memory banks, Unique ID, PC word, etc.

The format of the command is as follows:

```
TagInfo?
```

The reader finds a single tag and queries it for the information. It is best to confine your read zone, or create a mask on a specific tag when performing this command. The reader reports as much information as it can, depending on the tag type.

Alien Higgs3 & Higgs4:
TagInfo = M:xxxxxxxx KP:xx AP:xx EPC:xx,n USER:xx,n UPL:xxxx URL:xxxx PC:xxxx UID:xxxx xxxx xxxx xxxx

Alien Higgs2:
TagInfo = M:xxxxxxxx MAP:x KP:xx AP:xx EPC:xx,n User:xx PC:xxxx

Other Tags:
TagInfo = M:xxxxxxxx

Where,

M:xxxxxxxx is the first 2 words of the TID, indicating whether it is an EPCglobal tag, and the tag's manufacturer and model/version codes.

KP:xx is the lock state of the Kill Password
AP:xx is the lock state of the Access Password
EPC:xx,n is the lock state and length (in words) of the EPC bank
USER:xx,n is the lock state and length (in words) of the USER bank
UPL:xxxx are the User Block Permalock bits
URL:xxxx are the User Read Lock bits
PC:xxxx is the PC word
MAP:x is the Higgs2 memory map number
UID:xxxx xxxx xxxx xxxx are the four unique words guaranteed in every Higgs tag

TagInfo Examples	
Command Response	// Standard Higgs3 tag (6 EPC words, 32 USER words) >taginfo? TagInfo = M:E2003412 KP:UL AP:UL EPC:UL,6 USER:UL,32 UPL:0000 URL:0000 PC:3000 UID:012F F200 0044 F817
Command Response	// Reprogrammed Higgs3 tag (31 EPC words, 4 USER words) >taginfo? TagInfo = M:E2003412 KP:UL AP:UL EPC:UL,31 USER:UL,4 UPL:0000 URL:0000 PC:F800 UID:0133 F000 0486 6128
Command Response	// Higgs2 tag >taginfo? TagInfo = M:E2003411 MAP:4 KP:UL AP:UL EPC:UL,6 PC:3000
Command Response	// Non-Alien Higgs tag >taginfo? TagInfo = M:E2006004

G2Read

9900 | 9680 | 9650

The G2Read command performs a low-level memory read on a single Class 1/Gen 2 tag. You must specify the memory bank, word position into that bank, and the number of words to read. Some portions of tag memory may not be available to read. For example, an Access or Kill password field that is locked may not be read back.

The syntax of a G2Read command is:
G2Read = <bank> <wordPtr> <wordLen>
where:

<bank> = 0 – 3
<wordPtr> = position of first word to read (0-2097151, base 10)
<wordLen> = number of words to read (0-32, base 10)

The parameters you provide in the G2Read command may be separated by any combination of spaces or commas. The data returned is an even number of space-separated hexadecimal byte values upon success, or an error message.

The special case where <wordLen>=0 causes tags to return all of the memory in <bank> from <wordPtr> to the end of the bank. This is called a "zero-length read", and can be helpful if you don't know how large a particular memory bank is. If the amount of data returned is larger than 32 words, you may receive an error. In some situations, a zero-length G2Read may return tag data that you wouldn't normally see in a read with a non-zero length; if the tag happens to use "sliding partitions" in its memory implementation, then "the end of the bank" may not be clearly defined.

G2Read Examples	
	// Read the CRC and PC words (bank 1, words 0-1)
Command	>G2Read = 1 0 2
Response	G2Read = 42 E7 30 00 (CRC=42E7, PC=3000)
	// Read the 96-bit EPC (bank 1, starting at word #2, 6 words)
Command	>G2Read = 1 2 6
Response	G2Read = CF F7 CC 4D 22 26 FE BF 00 00 00 00
	// Compare to a "get TagList" response
Command	>get TagList
Response	Tag:CFF7 CC4D 2226 FEBF 0000 0000, Disc:2007/07/09 14:56:30...
	// Read two words of user memory (tag must support this)
Command	>G2Read = 3, 0, 2
Response	G2Read = DE AD BE EF
	// Read Access password (bank 0, words #2-3 - must not be locked)
Command	>G2Read = 0, 2, 2
Response	G2Read = 01 02 03 04
	// Try to read a memory location that doesn't exist
	// (tag not supporting User memory)
Command	>G2Read = 3 0 2
Response	Error 259: (Tag Error) Memory overrun or unsupported PC value.
	// Read all of the TID bank (zero-length read!)
Command	>G2Read = 2, 0, 0
Response	G2Read = E2 00 34 12 01 61 00 00 00 00 00 00 03 1D 01 60 70…

G2Write

9900 | 9680 | 9650

The G2Write command performs a low-level memory write to a single Class 1/Gen 2 tag. You must specify the memory bank, word position into that bank, and the data to write (an even number of bytes). Some portions of tag memory may not be available to write. For example, the CRC for a tag's EPC, or a locked field (usually) may not be written to. The syntax of a G2Write command is as follows:

```
G2Write = <bank> <wordPtr> <data>…
```

where: <bank> = 0-3
<wordPtr> = position of first word to write (0-2097151, base 10)
<data>… = an even number of bytes to write (1-32, base 10)

The parameters you provide in the G2Write command may be separated by any combination of spaces or commas. The reader responses with "Success!", or an error message.

G2Write Examples	
	// Write the EPC "the hard way" (bank 1, beginning at word #2)
Command	// This is **not** a good way to write the EPC!
Response	>G2Write = 1, 2, 11 22 33 44 55 66 77 88 99 10 11 12
	G2Write = Success!
Command	// Double-check with an inventory
Response	>get TagList
	Tag:1122 3344 5566 7788 9910 1112, Disc:2006/10/19 14:56:30...
	// Write to User memory (bank 3 - must be supported by the tag)
Command	>G2Write = 3, 0, DE AD BE EF CA FE
Response	G2Write = Success!
	// Read that User memory back (3 words)
Command	>G2Read = 3, 0, 3
Response	G2Read = DE AD BE EF CA FE
	// Try to write to a memory location that doesn't exist
	// (tag not supporting User memory)
Command	>G2Write = 3, 0, DE AD BE EF CA FE
Response	Error 259: (Tag Error) Memory overrun or unsupported PC value.

Programming an Entire Tag Image (Alien Higgs Tags Only)

The following commands expose custom Class 1/Gen 2 tag commands that are present only in the Alien Higgs2 and Higgs3 tags. Programming an Alien tag image writes all of the tag's memory (including passwords, EPC, User data, etc.) in one command. Programming an entire tag image is significantly faster than writing tag memory banks separately. In the current reader implementation, all of the fields/banks in the resulting tag are left unlocked.

The reader performs the ProgramAlienImage operation on the current ProgAntenna, and the tag must match the current acquire mask. Also, for Alien Higgs2 tags, the tag must still be in the factory-default "Higgs2_96" memory map, with no fields or banks locked. Alien Higgs3 tags do not have this restriction, and they also present a simplified memory map scheme.

ProgramAlienImage

9900 | 9680 | 9650

The ProgramAlienImage command writes the entire tag's memory map. No arguments are needed – the reader automatically constructs the appropriate sequence of image bytes to be sent to the reader by taking values from the ProgEPCData, ProgUserData, ProgG2AccessPwd, ProgG2KillPwd commands, along with information in the ProgAlienImageMap and ProgAlienImageNSI commands described later. Note that some of the data fields may be irrelevant, depending on the memory map selected.

The ProgEPCData and ProgUserData values may be automatically incremented following the ProgramAlienImage command, depending on the result of the operation and the setting of the ProgEPCDataInc and ProgUserDataInc properties. ProgUserData is only incremented if User data was written to the tag.

- The successful response to the ProgramAlienImage command is the complete image byte sequence sent to the tag.
- Alien Higgs2 tags only allow the ProgramAlienImage functionality if they are still in the factory default Higgs2_96 memory map, with no fields locked. You may change to one of the other mappings ONCE, or continue to rewrite the Higgs2_96 mapping.
- Alien Higgs3 tags may be reimaged as often as required.
- Alien Higgs4 tags do not support the ProgramAlienImage feature.
- If any fields in the tag are locked, they must be unlocked before writing an image map. In the current implementation, the tag remains completely unlocked after ProgramAlienImage. In the future, you may be able to specify desired lock states of various banks and memory blocks and have the ProgramAlienImage configure that as well.

ProgramAlienImage Example	
	// Preset the data to write (only needs to be done initially)
Command	Alien>ProgEPCData = 11 11 22 22 33 33 44 44 55 55 66 66
Response	ProgEPCData = 11 11 22 22 33 33 44 44 55 55 66 66
Command	Alien>ProgG2KillPwd = AA AA AA AA
Response	ProgG2KillPwd = AA AA AA AA
Command	Alien>ProgG2AccessPwd = BB BB BB BB
Response	ProgG2AccessPwd = BB BB BB BB
	// See following section for other ProgAlienImageMap examples
Command	Alien>ProgAlienImageMap = Higgs2_96
Response	ProgAlienImageMap = Higgs2_96
	// See following sections regarding ProgAlienImageNSI
Command	Alien>ProgAlienImageNSI = 00 00
Response	ProgAlienImageNSI = 00 00
	// Program the entire tag image
Command	Alien>ProgramAlienImage
Response	AAAA AAAA BBBB BBBB 1111 2222 3333 4444 5555 6666 3000 03B8
	// Verify the EPC
Command	Alien>t
Response	Tag:1111 2222 3333 4444 5555 6666, Disc:2007/07/06 13:05:34…
	// Verify the Kill and Access passwords
Command	Alien>G2Read = 0 0 4
Response	G2Read = AA AA AA AA BB BB BB BB
	// Check that the ProgEPCData field automatically incremented
Command	Alien>ProgEPCData?
Response	ProgEPCData = 11 11 22 22 33 33 44 44 55 55 66 67

ProgAlienImageMap

9900 | 9680 | 9650

Alien Class 1/Gen 2 tags may contain chips in a few varieties, Higgs2, Higgs3, and Higgs4. Only Higgs2 & Higgs3 support the ProgramAlienImage feature and its related properties.

HIGGS2

Alien Higgs2 tags can be configured with one of three memory maps. While the total amount of memory in the tag is constant, each memory map provides different combinations of EPC length and User memory options, at the expense of one or both of the Kill and Access passwords. The reader automatically constructs an image map specified by the ProgAlienImageMap property, and the relevant ProgEPCData, ProgUserData, etc. presets.

Higgs2 tags can be identified by finding 0x3411 in the 2nd word of the TID bank (bank 2):

```
Alien>G2Read = 2 1 1
G2Read = 34 11
```

Allowed values for ProgAlienImageMap when working with Higgs2 tags are:

ProgAlienImageMap (Higgs2)	EPC (bits)	USER (bits)	Access Pwd	Kill Pwd
Higgs2_96	96	0	✓	✓
Higgs2_128	128	0		✓
Higgs2_96U64	96	64		

- Alien Higgs2 tags only allow the ProgramAlienImage functionality if they are still in the factory default Higgs2_96 memory map, with no fields locked. You may change to one of the other mappings ONCE, or continue to rewrite the Higgs2_96 mapping. If any fields are locked, they must be unlocked before writing an image map.
- Previous firmware versions exposed these memory mappings as "EPC96", "EPC128", and "EPC96USER64". The reader still accepts those older memory map names, for backward compatibility.

HIGGS3

Alien Higgs3 tags have a much more flexible memory architecture. The Access and Kill passwords are always present, and there is always at least 96 bits of EPC memory and 64 bits of User memory. The partition between the EPC data and User data is not fixed, though, so you have some flexibility in how tag memory is divided between the two banks.

Higgs3 tags can be identified by finding 0x3412 in the 2nd word of the TID bank (bank 2):

```
Alien>G2Read = 2 1 1
G2Read = 34 12
```

Alien Higgs3 tags don't have fixed memory mappings like Alien Higgs2 tags do, so you change the partitioning of EPC+User memory by simply writing different amounts of data in each bank. To do a complete loading of Higgs3 tag memory, you select the "Higgs3" mapping in the ProgAlienImageMap command.

There is an alternative way to load Higgs3 memory which is significantly faster than the standard image load (which is, itself, significantly faster than writing each bank separately). This mechanism is known as "fast load" and writes Access & Kill passwords, and a 96-bit EPC only. Other EPC memory and all of the User memory is erased. You select this mode by choosing the "Higgs3_96" mapping in the ProgAlienImageMap command.

ProgAlienImageMap (Higgs3)	EPC (bits)	USER (bits)	Access Pwd	Kill Pwd
Higgs3	96-496	512-64	✓	✓
Higgs3_96	96	-	✓	✓

The total amount of memory allotted to the EPC+User banks cannot exceed 608 bits, and there are some word/block boundary conditions that have to be met as well. The allowed partitions are outlined below:

EPC Size (bits)	USER Max (bits)
0-96	512
112-160	448
176-224	384
240-288	320
304-352	256
368-416	192
432-480	128
496	64

ProgAlienImageMap Example – Higgs2 128-bit EPC	
	// Program a new Higgs2 tag with a 128-bit EPC
Command	>ProgAlienImageMap = Higgs2_128
Response	ProgAlienImageMap = Higgs2_128
Command	>ProgEPCData = 11 11 22 22 33 33 44 44 55 55 66 66 77 77 88 88
Response	ProgEPCData = 11 11 22 22 33 33 44 44 55 55 66 66 77 77 88 88
Command	>ProgG2KillPwd = AA AA AA AA
Response	ProgG2KillPwd = AA AA AA AA
	// Program the tag
Command	>ProgramAlienImage
Response	AAAA AAAA 1111 2222 3333 4444 5555 6666 7777 8888 4000 81B8
	// Verify the new 128-bit EPC
Command	>t
Response	Tag:1111 2222 3333 4444 5555 6666 7777 8888, Disc:2007/07/06…
	// Verify the Kill password
Command	>G2Read = 0 0 2
Response	G2Read = AA AA AA AA

ProgAlienImageMap Example – Higgs2 w/User Data	
	// Program a new Higgs2 tag with User memory
Command	>ProgAlienImageMap = Higgs2_96U64
Response	ProgAlienImageMap = Higgs2_96U64
Command	>ProgEPCData = 11 11 22 22 33 33 44 44 55 55 66 66
Response	ProgEPCData = 11 11 22 22 33 33 44 44 55 55 66 66
Command	>ProgUserData = AA AA BB BB CC CC DD DD
Response	ProgUserData = AA AA BB BB CC CC DD DD
	// Program the tag
Command	>ProgramAlienImage
Response	AAAA BBBB CCCC DDDD 1111 2222 3333 4444 5555 6666 3000 41B8
	// Verify the new EPC
Command	>t
Response	Tag:1111 2222 3333 4444 5555 6666, Disc:2007/07/06 14:14:24…
	// Verify the User data
Command	>G2Read = 3 0 4
Response	G2Read = AA AA BB BB CC CC DD DD

ProgAlienImageMap Example – Higgs3 Full Load	
	// Program a Higgs3 tag with a 496-bit EPC and 64-bits of User
Command	>ProgAlienImageMap = Higgs3
Response	ProgAlienImageMap = Higgs3
Command	>ProgEPCData = 11 12 13 14 15 16 17 18 21 22 23 24 25 26 27 28 31 32 33 34 35 36 37 38 41 42 43 44 45 46 47 48 51 52 53 54 55 56 57 58 61 62 63 64 65 66 67 68 71 72 73 74 75 76 77 78 81 82 83 84 85 86
Response	ProgEPCData = 11 12 13 14 15 16 17 18 21 22…
Command	>ProgUserData = A1 A2 A3 A4 A5 A6 A7 A8
Response	ProgUserData = A1 A2 A3 A4 A5 A6 A7 A8
Command	>ProgG2KillPwd = AA AA AA AA
Response	ProgG2KillPwd = AA AA AA AA
Command	>ProgG2AccessPwd = BB BB BB BB
Response	ProgG2AccessPwd = BB BB BB BB
	// Program the tag
Command	>ProgramAlienImage
Response	ProgramAlienImage = BBBB BBBB AAAA AAAA F800 1112 1314…
	// Verify the new 496-bit EPC
Command	>t
Response	Tag:1112 1314 1516 1718 2122 2324 2526 2728 3132 3334 3536 3738 4142 4344 4546 4748 5152 5354 5556 5758 6162 6364 6566 6768 7172 7374 7576 7778 8182 8384 8586, Disc:2008/08/12 13:43:16, Last:2008/08/12 13:43:16, Count:1, Ant:0, Proto:2
	// Verify the Kill & Access passwords
Command	>G2Read = 0 0 0
Response	G2Read = AA AA AA AA BB BB BB BB
	// Verify User memory
Command	>G2Read = 3 0 0
Response	G2Read = A1 A2 A3 A4 A5 A6 A7 A8

ProgAlienImageNSI

9900 | 9680 | 9650

The PC (Protocol Control) word of all Class 1/Gen 2 tags resides in bank #1 at word #1, just before the EPC data. The upper five bits of the PC word (bits 0-4, 0 being most significant) specify the length of the EPC data, and are automatically written by the reader when the EPC is programmed. The next two bits (bits 5,6) are designated Reserved for Future Use (RFU) by the EPC Class 1/Gen 2 protocol and should always be 0. The last nine bits (bits 7-15) of the PC word contain the Numbering System Identifier (NSI).

The first bit of the NSI (bit #7 of the PC word) is a toggle bit, indicating whether the EPC value is encoded using an EPCglobal™ numbering system (if bit #7 is a zero), or an ISO/IEC 15961 encoding (if bit #7 is a one). In the case of an EPCglobal encoding, the remainder of the NSI is reserved and should be all zeros. In the case of an ISO/IEC 15961 encoding, the remaining bits provide Application Family Identifier (AFI) information. The default value for the NSI is all zeros (indicating an EPCglobal encoding).

If you need to encode tags with a particular NSI value, you can provide this information with the ProgAlienImageNSI property. The reader automatically uses this value when constructing an Alien tag image map for you during execution of the ProgramAlienImage command. If you use the ProgramAlienImage command by manually providing the 24 image map bytes, you need to insert the desired NSI value into the image map data yourself, as part of the PC word.

The ProgAlienImageNSI is specified as one word (two bytes), although only the least significant 9 bits are actually used (the lower bit of the first byte, and all 8 bits of the second byte).

ProgAlienImageNSI Example

```
            // First, look at existing NSI (bank #1, word #1, ignore 1st nibble)
Command     Alien>G2Read = 1 1 1
Response    G2Read = 30 00
            // NSI is all zeros → EPCglobal encoding

Command     Alien>ProgAlienImageMap = Higgs2_96
Response    ProgAlienImageMap = Higgs_95

            // Program an NSI that is all ones
Command     Alien>ProgAlienImageNSI = 01 FF
Response    ProgAlienImageNSI = 01 FF

Command     Alien>ProgEPCData = 11 11 22 22 33 33 44 44 55 55 66 66
Response    ProgEPCData = 11 11 22 22 33 33 44 44 55 55 66 66

            // Program the tag
Command     Alien>ProgramAlienImage
Response    0000 0000 0000 0000 1111 2222 3333 4444 5555 6666 31FF 03B8

            // Verify the new custom NSI
Command     Alien>G2Read = 1 1 1
Response    G2Read = 31 FF
            // NSI is all ones
```

Programming Tags in AutoMode

Programming functions are fully supported by the reader's autonomous mode of operation. Using AutoMode, the user can assign the reader the task of programming, erasing, and locking tags. This is done by setting the AutoAction attribute to the appropriate action, as described below.

Some of the AutoMode functions, such as "ProgramEPC" and "ProgramAndLockEPC", require some additional values to be specified first, as described below.

AutoAction = ProgramEPC (was “Program”)

9900 | 9680 | 9650

When running in AutoMode with an AutoAction of ProgramEPC, the reader can automatically program a tag with a new EPC in AutoMode, and this action can be repeated indefinitely, each time incrementing the EPC to use for programming, so that groups of tags that were programmed while in AutoMode end up with sequential tag IDs.

To instruct AutoMode to program EPCs, issue the command:

```
AutoAction = ProgramEPC
```

Instead of "ProgramEPC", older readers used "Program", which is still accepted for backward compatibility.

Instead of reading tags during its "work" stage (refer to AutoMode state diagram in Chapter 2 for details), the reader programs the tag in the field with the EPC, specified by the ProgEPCData value (see above).

A successful programming event causes the evaluation stage of AutoMode to evaluate to `true`. A failed programming event evaluates to `false`.

The reader may optionally increment the next EPC to program, depending on the ProgEPCDataInc setting.

AutoAction = ProgramAndLockEPC

9900 | 9680 | 9650

When running AutoMode with an AutoAction of ProgramAndLockEPC, the reader can automatically program a tag with a new EPC and then lock it with a password. This event can be repeated indefinitely, each time incrementing the EPC by one.

To instruct AutoMode to program and lock EPCs, issue the following command

```
AutoAction = ProgramAndLockEPC
```

Instead of "ProgramAndLockEPC", older readers used "Program and Lock", which is still accepted for backward compatibility.

Instead of reading tags during its "work" stage (refer to AutoMode state diagram in Chapter 2 for details), the reader programs a tag with an EPC, as specified by the ProgEPCData value), and then locks it. For Class 1/Gen 2 tags, the ProgG2AccessPwd is written to the tag only if ProgG2LockType=Lock. For the PermaLock and PermaUnlock lock types, the ProgG2AccessPwd is not written (since the permanent nature of the lock doesn't require an Access password).

A successful ProgramAndLockEPC event causes the evaluation state of AutoMode to evaluate to `true`. A failed ProgramAndLockEPC event evaluates to `false`.

The reader may optionally increment the next EPC to program, depending on the ProgEPCDataInc setting.

AutoAction = ProgramUser

9900 | 9680 | 9650

When running in AutoMode with an AutoAction of ProgramUser, the reader can automatically program a tag with a new User data in AutoMode, and this action can be repeated indefinitely, each time incrementing the User data to use for programming, so that groups of tags that were programmed while in AutoMode end up with sequential User data.

To instruct AutoMode to program User data, issue the command:

```
AutoAction = ProgramUser
```

Instead of reading tags during its "work" stage (refer to AutoMode state diagram in Chapter 2 for details), the reader programs the tag in the field with the User data specified by ProgUserData value (see above).

A successful programming event causes the evaluation stage of AutoMode to evaluate to `true`. A failed programming event evaluates to `false`.

The reader may optionally increment the next User data to program, depending on the ProgUserDataInc setting.

AutoAction = ProgramAndLockUser

9900 | 9680 | 9650

When running AutoMode with an AutoAction of ProgramAndLockUser, the reader can automatically program a Class1/Gen2 tag with new User data and then lock it with a password. This event can be repeated indefinitely, each time incrementing the User data by one.

To instruct AutoMode to program and lock User data, issue the following command

```
AutoAction = ProgramAndLockUser
```

Instead of reading tags during its "work" stage (refer to AutoMode state diagram in Chapter 2 for details), the reader programs a tag with new User data specified by ProgUserData, and then locks it. The ProgG2AccessPwd is written to the tag only if ProgG2LockType=Lock. For the PermaLock and PermaUnlock lock types, the ProgG2AccessPwd is not written (since the permanent nature of the lock doesn't require an Access password).

A successful ProgramAndLockUser event causes the evaluation state of AutoMode to evaluate to `true`. A failed ProgramAndLockUser event evaluates to `false`.

The reader may optionally increment the next User data to program, depending on the ProgUserDataInc setting.

AutoAction = ProgramAlienImage

9900 | 9680 | 9650

When running in AutoMode with an AutoAction of ProgramAlienImage, the reader can automatically program an Alien Class 1/Gen 2 tag with a completely new image in AutoMode, and this action can be repeated indefinitely, each time incrementing the EPC (and User data, if relevant) to use for programming, so that groups of tags that were programmed while in AutoMode end up with sequential tag IDs.

To instruct AutoMode to program Alien G2 tags with new images, issue the command:

```
AutoAction = ProgramAlienImage
```

Instead of reading tags during its "work" stage (refer to AutoMode state diagram in Chapter 2 for details), the reader programs the tag in the field with a completely new memory image, as specified by the ProgAlienImageMap, ProgEPCData, ProgG2KillPwd, ProgG2AccessPwd, ProgUserData, and ProgAlienImageNSI values. Only those properties that are relevant to the selected ProgAlienImageMap are used.

A successful programming event causes the evaluation stage of AutoMode to evaluate to `true`. A failed programming event evaluates to `false`.

The reader may optionally increment the next EPC to program, depending on the ProgEPCDataInc setting. If the selected ProgAlienImageMap includes User data, then the reader may also increment the ProgUserData value, depending on the ProgUserDataInc setting.

AutoAction = Erase

9900 | 9680 | 9650

When running AutoMode with an AutoAction of Erase, the reader can automatically erase a tag in the field, and this action can be repeated indefinitely.

To instruct AutoMode to erase tags, issue the following command:

```
AutoAction = Erase
```

Instead of acquiring tags during its "work" stage (refer to AutoMode state diagram in Chapter 2 for details), the reader erases tags in the field.

A successful erase event in causes the evaluation stage of AutoMode to evaluate to `true`. A failed erase event evaluates to `false`.

Command	>NotifyNow
Response	Issuing Notify Trigger...

CHAPTER 6
LLRP

LLRP (Low Level Reader Protocol) is a specification for a standard network interface between RFID readers and controlling software or hardware. LLRP was created and ratified by EPCglobal in April, 2007, and is now at version 1.1. The goal of LLRP is to simplify application development for end-users, eliminating the need to conform to different proprietary communication protocols provided by each RFID vendor. There would no longer be a need to use different software "adapters", which would theoretically reduce the time needed to develop individual RFID solutions, and eliminate the risk associated with committing to one or two specific RFID vendors.

LLRP is supported on ALR-9900/+ readers only. It will not run on ALR-9680 or ALR-9650 readers at this time.

LLRP and ARP Coexistence

There are two end-points in the LLRP protocol – clients and readers. The client initiates a connection to a reader, and passes "specs" to it, detailing how the reader should be configured, air-protocol-specific properties, which antennas to use and when, and how the RFID data should be reported back to the client. LLRP also allows for "custom" specs, developed by individual reader vendors, to allow for proprietary operations on their particular reader models. This unfortunately negates the "write once, run anywhere" promise of a unifying standard.

While the LLRP service is running, the reader will listen on the LLRP TCP port (5084) for incoming LLRP connections, but will also continue to listen for Alien Reader Protocol (ARP) connections on its other communication channels – port 23, port 2300 (internal), and the serial interface. While LLRP is running, the reader will accept Alien commands sent over the ARP channels, but the Alien commands related directly to the radio subsystem will be blocked to prevent conflicts with the LLRP service. The Alien firmware is flexible enough to support both ARP and LLRP interfaces at the same time, so that you can still control the reader over ARP to, for example, start & stop services, change network settings, upgrade firmware, etc.

After stopping the LLRP service, we recommend you reboot the reader or perform a FactorySettings command.

Controlling the LLRP Service

You start, stop, enable, and disable the LLRP service on the reader using the Alien Reader Protocol's "service" command. The reader will only accept one LLRP connection at a time, but if a subsequent connection attempt is made, the reader will test the existing connection to see if it is still valid and if it is, it will reject the new connection. If the current connection is stale, it will disconnect the old connection and accept the new one.

Example:

```
// Check the status (stopped, autostart disabled, startup priority 15)
Alien>service llrp status
s d 15 llrp

// Start the LLRP service
Alien>service llrp start
R d 15 llrp

// While LLRP is running, can't read tags and do some other functions here
Alien>t
(No Tags)

Alien>BlinkLED = 0 255 100 5
Error 516: (Driver error) Some ARP commands are not available while LLRP is active.

// Force LLRP to disconnect any connected client
Alien>service llrp close
service llrp RUNNING

// Stop the LLRP service, fully reenabling ARP control of the reader
Alien>service llrp stop
s d 15 llrp

// Cause LLRP to automatically start when the reader boots
Alien>service llrp enable
s A 15 llrp

// Prevent LLRP from automatically starting when the reader boots
Alien>service llrp disable
s d 15 llrp
```

LLRP References

EPCglobal: LLRP Specification, v1.0.1
This is the first release of LLRP.
http://www.epcglobalinc.org/standards/llrp/llrp_1_0_1-standard-20070813.pdf

EPCglobal: LLRP Specification, v1.1.0
This is the newer release of LLRP, and supported by Alien LLRP.
http://www.epcglobalinc.org/standards/llrp/llrp_1_1-standard-20101013.pdf

EPCglobal: UHF Class 1, Gen 2 (C1G2) Standard
This is the C1G2 standard. You should be familiar with this before attempting LLRP.
http://www.epcglobalinc.org/standards/uhfc1g2/uhfc1g2_1_2_0-standard-20080511.pdf

LLRP Toolkit (LTK) Software Development Kit
LLRP Toolkit is a set of free SDKs in C, C++, .Net, Java, and Perl.
http://www.llrp.org/

LLRP Version 1.0.1 vs. Version 1.1.0

Most current LLRP implementations in use today implement the earlier v1.0 protocol. Alien firmware supports the newer LLRP v1.1 protocol.

LLRP v1.1 adds some new optional features, but most of the other changes involve additional parameters within certain specs, and expanded values for many enumerations of field values. For backward compatibility reasons, the reader answers all new connections in v1.0 mode. Once the client changes the protocol version to v1.1, then reader will operate under the v1.1 protocol for the duration of the connection. The protocol version can only be set once per connection.

VERSION NEGOTIATION

LLRP v1.1 introduces four new messages:

1. GET_SUPPORTED_VERSION
2. GET_SUPPORTED_VERSION_RESPONSE: Returns the currently-set protocol version, and the maximum version that the reader supports.
3. SET_PROTOCOL_VERSION: Includes the desired LLRP protocol version to be used from then on (may only be issued once per connection).
4. SET_PROTOCOL_VERSION_RESPONSE

NEW C1G2 FEATURES

Some new C1G2 v1.2.0 features are supported in LLRP v1.1, including BlockPermalock and tag recommissioning. Support for them is indicated in the C1G2LLRPCapabilities parameter, within a GET_READER_CAPABILITIES_RESPONSE message.

NEW COMMUNICATION STANDARDS

A new set of regional communication standards are now enumerated in the protocol.

ENHANCED REPORT TRIGGERS

There are two time-based report triggers that are new to v1.1.

5. Upon N seconds or end of AISpec
6. Upon N seconds or end of of ROSpec

LOOPSPECS

LLRP v1.1 defines a mechanism to loop execution of the Antenna Inventories within an ROSpec.

Alien LLRP Capabilities and Optional Features

Many aspects of the LLRP protocol are left up to the vendor to define or specify. The following table details the capabilities and optional features that the Alien LLRP implementation supports.

LLRP Feature or Capability	ALR-9900
General Device Capabilities	
MaxNumberOfAntennaSupported	4
CanSetAntennaProperties	-
HasUTCClockCapability	Y
# ReceiveSensitivityTableEntries	1
GPIOCapabilities	4 In, 8 Out
PerAntennaAirProtocols	Class1Gen2
LLRP Capabilities	
CanDoRFSurvey	-
CanReportBufferFillWarning	Y
SupportsClientRequestOpSpec	-
CanDoTagInventoryStateAwareSingulation	Y
SupportsEventAndReportHolding	Y
MaxPriorityLevelSupported	7
ClientRequestOpSpecTimeout	-
MaxNumROSpecs	64
MaxNumSpecsPerROSpec	16
MaxNumInventoryParameterSpecsPerAISpec	1
MaxNumAccessSpecs	256
MaxNumOpSpecsPerAccessSpec	8
Regulatory Capabilities	

# Transmit Power Levels	151
Min Transmit Power	16.6
Max Transmit Power	31.6
# RF Modes	5
C1G2 LLRP Capabilities	
CanSupportBlockErase	Y
CanSupportBlockWrite	Y
MaxNumSelectFiltersPerQuery	8
Custom Capabilities	
Separate Transmit Power when Writing	Y
Higgs Dynamic Authentication	Y
Hide User Blocks	Y

Alien LLRP RF Modes

The RF modes offered in the LLRP interface reflect the same RF Modes seen in the ARP interface. The following table lists these modes and their details.

Alien RF Modes	25M4 (DRM)	25FM0 (STD)	06FM0 (HS)	12M4	06M4
Mode Table Index (US-FCC)	**0**	**1**	**2**	**3**	**4**
Mode Identifier	**108**	**105**	**107**	**109**	**110**
DR Value	DRV_64_3	DRV_64_3	DRV_8	DRV_64_3	DRV_8
EPC HAG TC Conformance	0	0	0	0	0
M Value	MV_4	MV_FM0	MV_FM0	MV_4	MV_4
Forward Link Modulation	PR_ASK	PR_ASK	PR_ASK	PR_ASK	PR_ASK
Spectral Mask Indicator	DI	DI	MI	MI	MI
BDR Value	250000	250000	250000	250000	250000
PIE Value	1750	1750	1750	1750	1750
Min Tari Value	25000	25000	6250	12500	6250
Max Tari Value	25000	25000	6250	12500	6250
Step Tari Value	0	0	0	0	0

When setting the RF mode on an antenna, the LLRP specification is somewhat vague as to whether you give the zero-based Mode Table Index or the vendor-defined Mode Identifier, and vendor implementations differ in this regard. Alien accommodates this dual interpretation. You can either set a mode using its Mode Table Index, or the Mode Identifier.

For example, to set DRM mode, you can either use “1” (Mode Table Index) or “108” (Mode Identifier). The values of the Alien Mode Identifiers are specified in the table above.

NOTE: The entries in the RF Modes Table depend on the reader locality, so the same Mode Table Index value may represent different modes for different localities. The best way to determine the available modes and the Index range is to fetch the entire RF Mode table by reading the reader capabilities. The values of the Mode Table Index in the table above correspond to a US-FCC configured reader.

Alien LLRP Extensions

LLRP provides numerous "custom extension points" throughout the protocol, where vendors are allowed to insert customized parameters in the messages going to and from the reader, to provide functionality not specified in LLRP, and showcase unique features of their readers and tags. To some degree, this defeats the primary purpose of having a standardized protocol, so Alien has elected to focus on implementing the core protocol to its fullest (many ROSpecs & priorities, TagStateAware inventories, full support for masking, filters, and looping AISpecs), rather than inundating RFID users with new vendor-specific (and reader-specific!) options in the protocol.

That being said, there are some features and options that Alien felt were valuable enough to warrant deviating from the core standard. The Alien LLRP service supports the following three features, through LLRP extension parameters:

Alien Feature	Description
Dynamic Authentication	Performs additional verification steps to ensure the authenticity of Higgs3 & Higgs4 tags. Reports authentic-cation results and tag manufacturer information for all tags.
Write Power Control	Allows you to specify a different transmit power to use when performing write operations with the tag.
User Memory Read Locks	Allows you to hide individual blocks of User memory in Higgs3 & Higgs4 tags from being read.

Alien's unique manufacturer ID is 17996. This value and each custom parameters "type" value are required when constructing and inserting custom parameters into your LLRP messages.

Dynamic Authentication

Alien Higgs3 & Higgs4 tags have a feature that allows them to be validated for authenticity by the reader. The reader performs any additional authentication checks (on Higgs3/4 tags), and also reports back the manufacturer and tag model of every tag. You can make use of this feature in Alien LLRP by requesting the information via a custom parameter in the `ROReportSpec`, while you are configuring the rest of your `TagReportContentSelector`. The extra information will appear as a custom parameter in the `TagReportData` generated by the reader.

ALIENTAGREPORTCONTENTSELECTOR PARAMETER

The `AlienTagReportContentSelector` allows you to specify which Alien-only data fields to include in your `TagReportData`. Think of this as an extension of the `TagReportContentSelector` of the `ROReportSpec` where it is attached. The enableDynamicAuthentication field controls the dynamic authentication feature, and whether the tag manufacturer info will be included in the `TagReportData`.

The `AlienTagReportContentSelector` attaches to an `ROReportSpec`'s custom extension point.

AlienTagReportContentSelector (paramType #218)

EnableAlienDynamicAuthentication: Boolean

ALIENTAGREPORTDYNAMICAUTHENTICATION PARAMETER

When the reader collects tag authentication or manufacturer information, it attaches it to the `TagReportData` parameter as a custom parameter, alongside all of the other data for that tag.

AlienTagReportDynamicAuthentication (paramType #219)

TID: Unsigned Integer (4 bytes) – the first two words of the tag's TID; indicates whether it is an EPCglobal tag, and the tag's manufacturer and model number.

AuthenticateResult: Unsigned Char (1 byte)
Possible Values:

Value	Definition
0	Authentication Succeeded
1	Authentication Failed (possible fake!)
2	Alien Tag, but authentication not supported
3	Non-Alien Tag
4	Authentication Operation Failed (e.g. tag lost)

Write Power Control

Being able to use a different transmit power when writing to a tag, compared to reading tags, can be useful for confining the region of a tag-programming station. You could use a low transmit power when singulating a tag, to keep the read zone small, then switch to higher power when writing to the tag, to ensure write success.

ALIENWRITEPOWERCONTROL PARAMETER

The reader will automatically switch over to the power level given by the `AlienWritePowerControl` parameter when it is performing any type of write, lock, or kill operation on the tag. The value you specify is an index into the reader's transmit power level table, just like TransmitPower field in the `RFTransmitter` parameter.

The `AlienWritePowerControl` parameter attaches to the `C1G2AirProtocolInventoryCommand`'s custom extension point.

AlienWritePowerControl (paramType #100)

WriteTransmitPower: Unsigned Short Integer (2 bytes) – index into the Transmit Power Table.

User Read Locks

Align Higgs3 & Higgs4 tags allow you to hide individual blocks of User memory from others, unless they supply the correct access password. These "read locks" allow you to store sensitive data in the tag, which only you and trusted partners can read or modify. Higgs3 and Higgs4 tags subdivide their User memory into different sized blocks, for read lock purposes. Higgs3 uses blocks that are four words (64 bits total) in length, and Higgs4 uses blocks that are 2 words (32 bits) in length. When you set read locks on the User bank, you set all of the lock states for all of the blocks at the same time; there is no way to only change one block's status without also re-asserting the other statuses of the other blocks.

ALIENHIDEBLOCKS PARAMETER

The `AlienHideBlocks` parameter acts just like other `C1G2OpSpec`s in the protocol, and it is attached to a specific `AccessSpec`. When a tag that matches the `AccessSpec`'s `C1G2TagSpec` parameter, then all of the `OpSpecs` defined in that `AccessSpec` are run, including the `AlienHideBlocks` parameter. The `AlienHideBlocks` parameter contains an OpSpecID field, so that you can identify the results of it's action in the generated `TagReportData`. It also contains a field for the memory bank to act on (only bank 3 is supported), the necessary Access Password, and the bytes containing the lock states.

Each of the blocks in the User memory of Higgs3/4 tags are numbered sequentially, and have a corresponding bit in the ReadLockMask that you use to set the lock states. The first block corresponds to the first (highest order) bit in the mask, followed by the second block in the next-higher bit, etc. Since the ReadLockMask is 8 bits long, if you wanted to hide only the first block of User memory, you would use a value of 0x80 (binary: 10000000), ***not*** 0x01 (binary: 00000001).

The `AlienHideBlocks` custom parameter attaches to either the `AccessCommand`'s custom extension point (LLRP v1.0 or 1.1), or directly onto the `AccessCommand`'s list of `OpSpec`s (LLRP v1.1 only).

AlienHideBlocks (paramType #300)

OpSpecId: Unsigned Short Integer (2 bytes)

Bank: Unsigned Char (1 byte) – currently, only bank 3 is supported.

AccessPassword: Unsigned Integer (4 bytes)

ReadLockMask: Unsigned Char (1 byte)

ALIENHIDEBLOCKSRESULT PARAMETER

When an `AccessSpec` is run containing an `AlienHideBlocks` parameter, the results of the read-lock operation will be in a custom parameter in that tag's `TagReportData`, along with the other data gleaned from the tag during the inventory, and the other `OpSpecResult`s that may have been collected as well.

The `AlienHideBlocksResult` parameter attaches to the `TagReportData`'s custom extension point.

AlienHideBlocksResult (paramType #301)

OpSpecId: Unsigned Short Integer (2 bytes)

Result: Unsigned Char (1 byte)
Possible Values:

Value	Definition
0	Succeeded.
1	Insufficient power at the tag.
2	Non-specific tag error.
3	No response from tag.
4	Non-specific reader error.
5	Incorrect password.
6	Tag memory overrun.
7	Tag memory locked.

APPENDIX A
DTDs for XML Data Structures

The reader has the capability to generate three different types of XML-formatted documents. This appendix gives a Document Type Definition (DTD) for each of these XML documents.

Heartbeat DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Alien-RFID-Reader-Heartbeat [
  <!ELEMENT Alien-RFID-Reader-Heartbeat (ReaderName, ReaderType,
            IPAddress, CommandPort, HeartbeatTime, MACAddress?, ReaderVersion?)>
    <!ELEMENT ReaderName    (#PCDATA)>
    <!ELEMENT ReaderType    (#PCDATA)>
    <!ELEMENT IPAddress     (#PCDATA)>
    <!ELEMENT CommandPort   (#PCDATA)>
    <!ELEMENT HeartbeatTime (#PCDATA)>
    <!ELEMENT MACAddress    (#PCDATA)>
    <!ELEMENT ReaderVersion (#PCDATA)>
]>
```

TagList DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Alien-RFID-Tag_List [
  <!ELEMENT Alien-RFID-Tag_List (Alien-RFID-Tag*)>
    <!ELEMENT Alien-RFID-Tag (TagID, DiscoveryTime, LastSeenTime,
              Antenna, ReadCount, Protocol?, D1?, D2?, D3?, D4?)>
      <!ELEMENT TagID         (#PCDATA)>
      <!ELEMENT DiscoveryTime (#PCDATA)>
      <!ELEMENT LastSeenTime  (#PCDATA)>
      <!ELEMENT Antenna       (#PCDATA)>
      <!ELEMENT ReadCount     (#PCDATA)>
      <!ELEMENT Protocol      (#PCDATA)>
      <!ELEMENT D1            (#PCDATA)>
      <!ELEMENT D2            (#PCDATA)>
      <!ELEMENT D3            (#PCDATA)>
      <!ELEMENT D4            (#PCDATA)>
]>
```

Notification DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Alien-RFID-Reader-Auto-Notification [
  <!ELEMENT Alien-RFID-Reader-Auto-Notification (ReaderName, ReaderType,
            IPAddress, CommandPort, Time, Reason, StartTriggerLines?,
            StopTriggerLines?, Alien-RFID-Tag-List)>
    <!ELEMENT ReaderName        (#PCDATA)>
    <!ELEMENT ReaderType        (#PCDATA)>
    <!ELEMENT IPAddress         (#PCDATA)>
    <!ELEMENT CommandPort       (#PCDATA)>
    <!ELEMENT MACAddress        (#PCDATA)>
    <!ELEMENT Time              (#PCDATA)>
    <!ELEMENT Reason            (#PCDATA)>
    <!ELEMENT StartTriggerLines (#PCDATA)>
    <!ELEMENT StopTriggerLines  (#PCDATA)>
]>
```

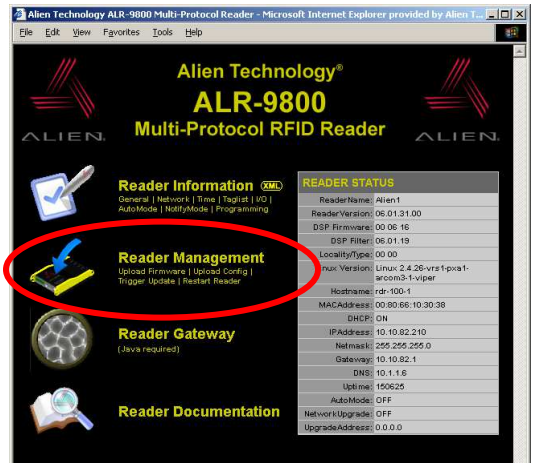
APPENDIX B
Upgrading Reader Firmware (GUI)

Periodically, Alien Technology provides improvements and bug fixes to the firmware running inside the reader. These improvements are distributed as a firmware upgrade file, which is uploaded to the reader either via a web interface to the reader, or using the Gateway Demonstration Software.

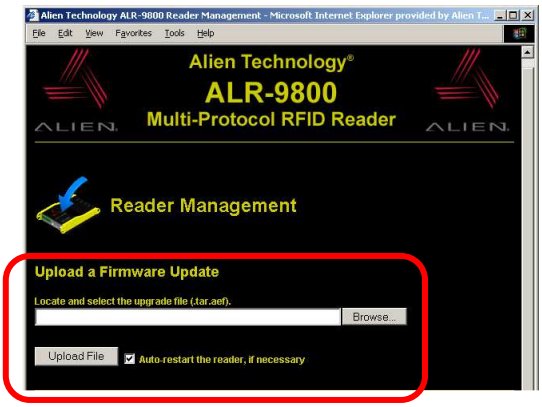
Performing a firmware upgrade typically resets the reader's configuration to a default state, so care should be taken to record the reader's state before performing the upgrade, then issuing the necessary commands to return the reader to its pre-upgrade state. Network settings such as DHCP, IP Address, etc. are not affected.

The current series of readers can be upgraded over the network using the Gateway Demonstration Software. Readers must have firmware later that 05.10.01 in order to use the Gateway for upgrades. Current readers do not support upgrading through the serial interface.

These readers can also be upgraded using the reader's built-in web interface. Browse to the reader's web server and navigate to the Reader Management page. You need to enter your TCP login username and password ("alien" / "password" by default) to access the Reader Management page.



Using the form on that page, select the upgrade file and submit the form.



Readers can also be configured to fetch their own updates when they are starting up. See the section titled, "Pulling Upgrades", in the next appendix.

APPENDIX C
Upgrading Reader Firmware (Programmatic)

This appendix describes how to push upgrades to an reader, using the standard HTTP POST protocol, as well as how to configure the reader to automatically pull upgrades down from an FTP or HTTP server.

File Structure of an Upgrade

The typical upgrade file that is provided to customers is an encrypted file. The filenames have the following format:

ALRx800_fw_050921.tar.aef

The first portion, "ALRx800" identifies the target reader platform. It may appear as "ALRx800" even if it applies to the entire model line, since current readers are all descended from the ALR-9800 platform and use the same firmware builds.

The second portion of the filename, "fw", indicates the component in the reader that is being upgraded ("firmware").

The third portion is the date of the upgrade, in YYMMDD format. It typically matches the YY.MM.DD value given by the ReaderVersion command.

Finally, the two extensions indicate that the package is a tarball (.tar), and that it has been encrypted using Alien's own encryption algorithm (.aef = Alien Encrypted File).

Pushing Upgrades

Appendix B already demonstrated how to push upgrades to the reader, using the reader's web interface. This same interface is available for use by programmers, using the standard HTTP POST mechanism. Reader firmware upgrades as well as reader macro files (.arm) can all be uploaded to the reader through this mechanism.

HTTP Post Request

Uploading a file to the reader takes nothing more than issuing a POST request to the reader's web server, at the URL:

http://<reader's IP>/cgi-bin/upgrade.cgi

When issuing the request, the following requirements must be adhered to. Implementations vary in how these requirements are actually accomplished. An example program written in Java is provided for reference.

- The request method must be POST
- Content-Type for the request must be "multipart/form-data"
- Authentication for the request must be "Basic <BASE64-encoded user:pass>"
- Content-Disposition for the file attachment = "form-data"
- Name for the file attachment = "uploadedFile"
- Filename for the file attachment = <localFilePath>
- Content-Type for the file attachment = "text/plain"

HTTP Post Response

After posting the file to the upgrade CGI, the reader responds with a series of exit codes and the final disposition of the reader after the upgrade is installed. Each line of the response is terminated with a single newline character.

The first line of the response indicates the result of the file upload itself. If the file was received and successfully written to the reader's filesystem, then a "0" (zero) character is returned, otherwise a number greater than "0", representing the error code, is returned. Note that these are ASCII characters, not the byte values, 0x00, 0x01, etc.

If an error is encountered while receiving the file, then the second line of the response is a string describing the nature of the problem, and this concludes the reader's response.

If the upload is successful, the reader attempts to unpack and install the upgrade, and the second line of the response indicates the result of this step. The result code returned is one of the following:

0 – Upgrade successful; reboot required.
1-127 – Upgrade not successful.
-256 – Upgrade successful; no reboot is required.

In the case of an unsuccessful unpacking or installation, then the third line of the reader's response is a string describing the nature of the problem. Otherwise, the reader returns one of the following:

Done – Upgrade complete; no reboot required.
Rebooting – Upgrade complete; reader is rebooting

Finally, the reader terminates its response with a single null (0x00) value.

Sample Java Implementation

```
import java.net.*;
import java.io.*;

public class HTTPFilePost {
  private static final String READER_URL = "http://192.168.1.100/cgi-bin/upgrade.cgi";
  private static final String FILE_PATH = "C:\\9800upgrades\\ALR9800_fw_test.tar.aef";
  private static final String CONTENT_BOUNDARY = "pflxm";
  private static final String EOL = "\r\n";
  private static final String username = "alien";
  private static final String password = "password";

  public static final void main(String args[]) {
    FileInputStream fileInputStream = null;
    OutputStream outputToReader = null;
    BufferedReader inputFromReader = null;

    try {
      // Open the connection
      URL testPost = new URL(READER_URL);
      URLConnection conn = testPost.openConnection();

      // Deal with authentication
      String userpass = username + ":" + password;
      String encoding = new sun.misc.BASE64Encoder().encode(userpass.getBytes());
      conn.setRequestProperty("Authorization", "Basic " + encoding);
```

```
      // This makes it a POST request
      conn.setDoOutput(true);
      conn.setRequestProperty("Content-Type",
                              "multipart/form-data; boundary=" + CONTENT_BOUNDARY);
      outputToReader = conn.getOutputStream();

      // Setup the file and start the http request
      File uploadFile = new File(FILE_PATH);
      fileInputStream = new FileInputStream(uploadFile);
      String requestHeader = "--" + CONTENT_BOUNDARY + EOL
                           + "Content-Disposition: form-data; name=\"uploadedFile\";"
                           + " filename=\"" + FILE_PATH + "\"" + EOL
                           + "Content-Type: text/plain" + EOL + EOL;
      System.out.println("Request Header:\n" + requestHeader);
      outputToReader.write(requestHeader.getBytes());

      // Read from the upgrade file and write it to the reader
      byte[] fileBuffer = new byte[512];
      int totalBytesRead = 0;
      int numBytesRead = fileInputStream.read(fileBuffer, 0, 512);
      while (numBytesRead != -1) {
        totalBytesRead += numBytesRead;
        outputToReader.write(fileBuffer, 0, numBytesRead);
        numBytesRead = fileInputStream.read(fileBuffer, 0, 512);
      }
      System.out.println("Wrote " + totalBytesRead + " bytes from the upgrade file.\n");

      // Close the local upgrade file and finish the POST request
      fileInputStream.close();
      fileInputStream = null;
      String requestFooter = EOL + "--" + CONTENT_BOUNDARY + "--" + EOL;
      System.out.println("Request Footer:\n" + requestFooter);
      outputToReader.write(requestFooter.getBytes());

      // Construct the BufferedReader to get the reader's response
      inputFromReader = new BufferedReader(new InputStreamReader(conn.getInputStream()));

      // Read back the server's response
      String responseLine = inputFromReader.readLine();
      while (responseLine != null) {
        System.out.println("Reader Response: " + responseLine);
        responseLine = inputFromReader.readLine();
      }

      // Close everything
      inputFromReader.close();
      outputToReader.close();
      inputFromReader = null;
      outputToReader = null;
    } catch (Exception ex) {
      ex.printStackTrace();
    } finally {
      if (fileInputStream != null) {
        try { fileInputStream.close(); } catch (Exception ignore_it) {}
      }
      if (inputFromReader != null) {
        try { inputFromReader.close(); } catch (Exception ignore_it) {}
      }
```

```
      if (outputToReader != null) {
        try { outputToReader.close(); } catch (Exception ignore_it) {}
      }
    }
  }

} // End of class HTTPFilePost
```

Sample output from this program:

```
Request Header:
--pflxm

Content-Disposition: form-data; name="uploadedFile"; filename =
"C:\9800upgrades\ALR9800_fw_test.tar.aef"
Content-Type: text/plain

Wrote 39 bytes from the upgrade file.

Request Footer:
--pflxm--

Reader Response: 0
Reader Response: 0
Reader Response: Rebooting
Reader Response: [null]
```

Pulling Upgrades

The reader can be configured to connect to an upgrade host and download and install updates on its own. The only requirement is an HTTP or FTP server that the reader can access to look for updates. HTTPS (secure HTTP) is also supported.

As opposed to the "push" model described previously (where the host initiates a connection to the reader and uploads an upgrade file), this "pull" model involves a passive host that the reader initiates a connection to and downloads an upgrade file from.

Configuring the Reader

There are two properties that need to be configured on the reader in order to facilitate pulling upgrades.

UPGRADEADDRESS

The UpgradeAddress specifies the URL of the directory where the upgrade file can be accessed. Both the HTTP and FTP protocols are supported, as well as HTTPS, the secure version of HTTP. Since it is a directory we are specifying, the URLs should all end in a slash. For example:

```
Alien>UpgradeAddress = http://192.168.1.1/<path>/
UpgradeAddress = http://192.168.1.1/<path>/

Alien>UpgradeAddress = ftp://ftp.mydomain.com/<path>/
UpgradeAddress = ftp://ftp.mydomain.com/<path>/

Alien>UpgradeAddress = https://secureHTTPhost/<path>/
UpgradeAddress = https://secureHTTPhost/<path>/
```

SPECIFYING USERNAME AND PASSWORD

For the protocols that require authentication, the username and password are prepended to the domain portion of the UpgradeAddress URL, with a colon separating the username from the password, and an "@" symbol separating them from the domain. For example:

```
Alien>UpgradeAddress = ftp://user:password@ftpserver/<path>/
UpgradeAddress = ftp://user:password@ftpserver/<path>/
```

Enabling Upgrade Pulls

To turn on the upgrade pull mechanism, simply set the NetworkUpgrade property to on.

```
Alien>NetworkUpgrade = On
NetworkUpgrade = On
```

Make sure to issue the "save" command after configuring the reader, in order to write the new settings to flash. Otherwise, when the reader reboots, these settings revert back to their previous values.

Setting up the Upgrade Host

The upgrade host must be running an appropriate service corresponding to the protocol specified in the reader's UpgradeAddress property.

Place the upgrade file you wish to make available to the readers within the upgrade directory pointed to by the UpgradeAddress. There can be other files in the directory as well. The particular .tar.aef file that is pulled down by the reader when it upgrades itself must be listed in a file name "control", which must also reside in the same upgrade directory.

NETWORKUPGRADE CONTROL FILE

Create a file called "control" inside the upgrade directory. If your server balks at serving files without extensions, you can optionally name the file "control.txt". This file lists the filenames of the particular upgrade files that the reader downloads (typically, though, there will only be a single file for upgrading). Each line listing the upgrade files should be terminated with a single linefeed character ("\n" = 0x0A).

This control file allows you to have multiple versions of reader software on-hand in the upgrade directory, and control which one is made active by simply changing the entry in the control file. It also ensures that readers are always pointing at an accessible file, and when a new release is made available, a single change at the network host enables access to the new file for all configured readers. Readers don't have to know the particular name of the new upgrade – just where the control file is located.

Control files may also point to reader macros (files ending in .arm). These files are downloaded and installed in the reader, replacing any macros that may exist with the same names.

All of the firmware files referenced in a control file must be in the same directory as the control file. Pointing to other directories, for instance "../ALR-x800_fw_070131.tar.aef" or "/alienFirmware/ALR-x800_fw_070131.tar.aef", is not supported at this time.

How It Works

The reader runs a script that handles pulling upgrade files down from the server and installing them. It runs this script automatically when the reader boots up, but it can also be triggered through the web interface or with the UpgradeNow command.

The update script first checks to see if the NetworkUpgrade property is set to "On". If it isn't, the script exits and the Alien reader subsystem starts up as it normally would.

If the NetworkUpgrade flag is "On", the reader attempts to download the control file located at the URL specified by the UpgradeAddress property. If no control file is found, the script exits and the Alien reader subsystem starts up as normal.

If the control file exists, the script looks inside it for a filename, and examines the filename for versioning information. For example, a typical upgrade file named, "ALR9800_fw_060217.tar.aef", corresponds to a ReaderVersion of 06.02.17. The script compares this version to what is currently installed in the reader, and if it is *different*, it downloads the upgrade, unpacks it, and installs it.

Note that this allows the reader to be back-revved. As long as the upgrade version is different from what is installed, the upgrade (or downgrade) takes place.

APPENDIX D
EN 300 220 Duty Cycling

Operation of an Alien -EMA reader under EN 300 220 requires the reader to maintain a 10% duty cycle for RF emission.

Under normal operating conditions, this requirement may be met if the reader is polled in synchronous mode comparatively infrequently, by using IO-triggered operation with AutoMode, or by configuring AutoMode to with continuous duty cycling. This appendix documents a way to run the reader in AutoMode implementing continuous duty cycling.

Duty Cycle Control in AutoMode

The relative amount of time a reader spends on or off can be controlled by using the Appropriate AutoStopTimer, AutoTruePause and AutoFalsePause values.
To achieve a 10% duty cycle (where the reader is on for 10% of the time), one could set the following AutoMode parameters:

AutoStopTimer	= 100	(run for ~100ms before evaluating)
AutoTruePause	= 900	
AutoFalsePause	= 900	(set the same as AutoTruePause)

Multiples of these values (500/4500/4500 or 1000/9000/9000) may also be used.

APPENDIX E
EN 302 208 v.1.3.1 Operation

The ALR-9900+ and ALR-9680 readers configured for operation in European Union (ALR-9900-EMA and ALR-9680-EMA) implement the EN 302 208 v.1.3.1 specification. The default mode of operation is hopping between 4 channels spaced at 600 kHz.

These readers can also operate in a fixed frequency mode with a maximum continuous transmit time of 4 seconds before pausing for 100ms before transmitting again on the same channel.

Setting Fixed Frequency Operation (ALR-9900/9680-EMA Only)

To make the reader operate in either the default hopping mode or on a single fixed frequency channel use the following command:

```
RFChannel = <channel>
```

- Allowed Values: `0-3` or `-1` (hopping)
- Default Value: `-1` (hopping)
- The 100ms channel dwell time is controlled by the reader. There is no need to manually duty cycle the reader for this mode of operation.

RFChannel Examples	
Command Response	Alien>**RFChannel?** RFChannel = -1 // hopping
Command Response	// set fixed frequency channel 1 Alien>**RFChannel** = 1 RFChannel = 1

APPENDIX F
Error Codes

The following tables list the error codes and error messages generated by the reader, and brief descriptions of how and why they are generated.

General Errors (1-49)

Error #	Error Message Details
1	**Command not understood.** A command was issued which the reader does not understand.
4	**Invalid use of command.** A valid command is being used improperly (changing a read-only command, no "get" or "set" when required, etc.)
5	**Timeout waiting for user command.** The user started typing a command, and didn't finish within 60 seconds.
7	**Command requires a value in the format Command=Value.** A command requiring a value was not provided a value.
10	**Value out of range.** An argument was supplied that was not in the valid range (string length, non-integer when integer expected, outside min/max bounds).
19	**Invalid Username and/or Password.** A TCP login attempts failed the username/password check.
22	**Invalid IP address.** A supplied IP address argument doesn't look like a valid IP address.
24	**Invalid format.** The structure of the supplied arguments is wrong.
27	**Invalid context.** Selected tag protocol doesn't support a data size (e.g. EPC length) or command (e.g. Unlock when ProgProtocol != 2).
28	**Unknown error.** (internal errors)
29	**Parameter required.** A notification was attempted, but no NotifyAddress was specified.
34	**Invalid mask.** bitPtr + bitLen larger than EPC memory, or bitLen doesn't match provided mask bytes.
36	**Unable to deliver notification.** A problem was encountered trying to send a notification message.
37	**Input buffer overflow.** Too many characters were entered on a single command line.
39	**Timeout waiting for command response.** A command was issued to an internal reader component and no reply was received after 60 seconds.
38	**Invalid parameter.** A parameter to a command is invalid – a macro name, with no corresponding macro, for instance.

Macro Errors (50-60)

Error #	Error Message Details
51	**Macro name is missing.** The command requires a macro name, and it is missing.
53	**Macro name is too long.** The macro name exceeds 64 characters in length.
54	**Macro does not exist.** You are trying to run or delete a macro that can't be found.
55	**The supplied macro is empty.** You are trying to run a macro that has no commands.
60	**Macro access error.** You are trying to run or delete a macro, and the reader does not have necessary filesystem permissions.

DSP Errors (128-255)

Error (dec)	Error (hex)	Error Message Details
133	0x85	**Memory overrun.** A non-existent memory location was accessed.
134	0x86	**No tag found.** A program operation was attempted, and no tag was found in the field.
135	0x87	**Erase failed.** An attempt to erase a tag failed.
136	0x88	**Program failed.** An attempt to program data into a tag failed to write.
137	0x89	**Tag is locked.** An attempt was made to program locked tag data.
138	0x8A	**Kill failed. Check the kill password.** An attempt to kill a tag failed. Either the tag cannot be killed, or the wrong password was used.
139	0x8B	**Lock attempt failure.** An attempt to lock a tag's data failed.
140	0x8C	**Tag data memory size mismatch.** An incorrect amount of data was attempted to be written.
141	0x8D	**Hardware error.** Trouble accessing a component of the radio hardware.
146	0x92	**Lock failed CRC error.** The tag failed a CRC check after locking.
147	0x93	**A previously found tag is not responding to program operations.**
148	0x94	**Kill Pwd could not be verified during Lock, no Lock performed.** During a C1 lock operation, the Kill Pwd failed to be written.
149	0x95	**Singulation error** Multiple tags responded to a command.

154	**0x9A**	**Read error.** Unknown tag error.
157	**0x9D**	**No tag found matching the provided EPC.** EPC specified in a Kill operation doesn't any tags in the field.
158	**0x9E**	**Access failed.** The provided AcqG2AccessPwd doesn't match the tag's Access Pwd.
159	**0x9F**	**Malformed Tag response on write.** The tag's asynchronous response to a write operation could not be decoded.

G2 Tag Errors (256-511)

These are the raw error codes that are returned by a G2 tag during either one of the low-level G2Read and G2Write commands.

Error #	Error Message Details
256	**(Tag Error) Other error.** G2 tag error not covered by other error codes.
259	**(Tag Error) Memory overrun or unsupported PC value.** A non-existent memory location was accessed, or the desired PC value is not supported by the tag.
260	**(Tag Error) Memory locked.** An attempt was made to access locked tag memory.
267	**(Tag Error) Insufficient power.** The tag has insufficient power to perform a memory write operation.
271	**(Tag Error) Non-specific error.** Tag error (when error-specific codes are not supported by the tag).