*Thinkify, LLC*

# TR100 RFID Reader Module

***Setup Guide and Protocol Reference***

***January 2021***

# Notices

## *Note Regarding RF Exposure*

This equipment complies with FCC radiation exposure limits set forth for an uncontrolled environment.  This equipment should be installed and operated with minimum distance of 20 cm in the US or 33cm, (Canada), between the radiator (antenna) and your body.  This transmitter must not be co-located or operating in conjunction with any other antenna or transmitter.

## ***FCC and IC Notice and Cautions:***

### **FCC**

Any changes or modifications to this device not expressly approved by Thinkify, LLC could void the user's authority to operate the equipment.

This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (1) this device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio/TV technician for help.

### **IC**

This device contains licence-exempt transmitter(s)/receiver(s) that comply with Innovation, Science and Economic Development Canada’s licence-exempt RSS(s). Operation is subject to the following two conditions:

1. This device may not cause interference.
2. This device must accept any interference, including interference that may cause undesired operation of the device.

## *Modular Use, Procedures, and Integration Instructions*

The Thinkify module must only be used under the following conditions outlined throughout this document.

### Voltage:

The unit should have a stable and lower noise 5 volt supply from either the USB or the serial interface. Noise should be less than 50 mV peak to peak and the 5 Volts supply should vary less than plus or minus 5% under operation.

### Heat:

The unit should be placed in such a manner that heat is allowed to exit the module. It is recommended that the device be placed with at least 200 square cm of heat sink when operating near the upper power, (above 200 mW), of the unit.

### Antennas:

Only patch antennas of 6 dBil or less gain are authorized with this module. In all cases radiated power should be kept to a minimum. All end users shall adhere to all aspects of the integration procedures and instructions outlined in this section. This radio transmitter 9962A-100 has been approved by Innovation, Science and Economic Development Canada to operate with the antenna types listed below, with the maximum permissible gain indicated. Antenna types not included in this list that have a gain greater than the maximum gain indicated for any type listed are strictly prohibited for use with this device.

Acceptable antennas, US and Canada:

Patch antennas only. All patch antennas must have less than 6dBil gain. All must be nominally of 50 Ohm impedance.

No other antenna types are approved at this time.

Please contact Thinkify in reference to any antennas used.

### Separation:

This equipment should be installed and operated with minimum distance of 20 cm in the US or 33cm, (Canada), between the radiator (antenna) and any part of any human body. The end user must include instructions that notify all users to keep a minimum distance as above.

**Sole Transmitter:**

This module is to be used alone for the purposes of RFID. This transmitter must not be co-located or operating in conjunction with any other antenna or transmitter. If there are other transmitters co-operational in the immediate vicinity of the TR100 module or if there are are any questions about co-operational transmitters please contact Thinkify.

**Equipment Authorization:**

The end user is responsible for obtaining any and all required equipment authorization for the applicable unintentional radiator functions (part 15 sub-part B), of the host product.

**Testing:**

The end user will perform testing on the host product with the TR100 module operating to confirm that the final product meets all FCC requirements. Testing requirements are further defined in the following:

https://apps.fcc.gov/kdb/GetAttachment.html?id=bNCiEdkFEKnHsZF9GHCNdg%3D
%3D&desc=996369%20D04%20Module%20Integration%20Guide
%20V02&tracking_number=44637

**Label:**

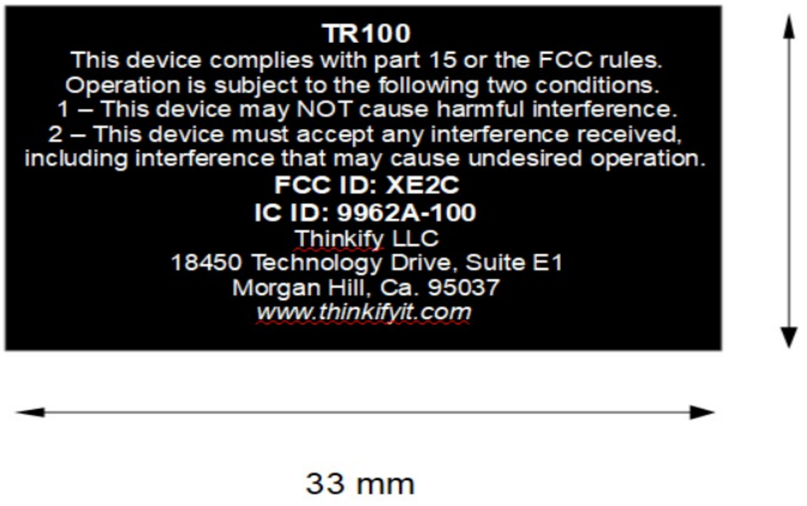
The Thinkify label should not be removed. End users are responsible for the continued compliance of their modules to the FCC and IC rules. Host product manufacturers must provide a physical or e-labelstating"Contains FCC ID" with their finished product. See Guidelines for Labeling and User Information for RF Devices in KDB Publication 784748. The module is labeled with the following identification:

**FCC ID: XE2C**
**IC: 9962A-100**

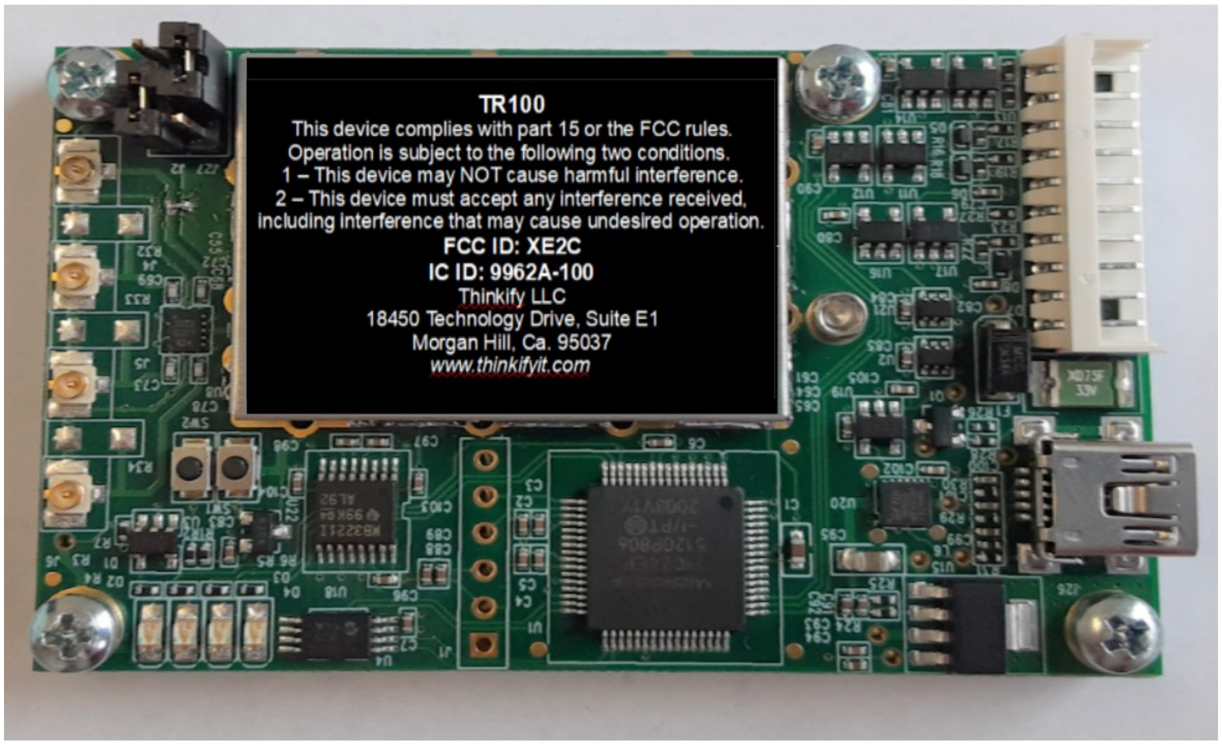
The host device in which the module is integrated into should be labeled with the following identification:

**Contains FCC ID: XE2C**
**Contains IC: 9962A-100**

Label:



**TR100**
This device complies with part 15 or the FCC rules.
Operation is subject to the following two conditions.
1 – This device may NOT cause harmful interference.
2 – This device must accept any interference received, including interference that may cause undesired operation.
**FCC ID: XE2C**
**IC ID: 9962A-100**
Thinkify LLC
18450 Technology Drive, Suite E1
Morgan Hill, Ca. 95037
*www.thinkifyit.com*



**TR100**
This device complies with part 15 or the FCC rules.
Operation is subject to the following two conditions.
1 – This device may NOT cause harmful interference.
2 – This device must accept any interference received, including interference that may cause undesired operation.
**FCC ID: XE2C**
**IC ID: 9962A-100**
Thinkify LLC
18450 Technology Drive, Suite E1
Morgan Hill, Ca. 95037
*www.thinkifyit.com*

Label:

**Test Modulation Modes:**

Test modulation modes, to insure that the TR100 when included within the host device remains compliant, are available from Thinkify upon request.

## *Revision History*

7 October 2020 - drafted the TR100 Setup and User's Manual.

7 January 2021 – removed outdated output power table

27 January 2021 – added required FCC and IC integration instructions

## About Thinkify, LLC

Thinkify, LLC is a wireless technology company specializing in RFID hardware and software products. With 30 years of combined experience in RFID and over 35 patents in the field, our founding team is one of the technically strongest in the industry.

Our focus is *embedded* RFID. -- Applications where we use RFID to enable common objects, devices and whole environments to become aware of the world around them. This capability can transform the way people and objects interact, blurring the line between the physical world and the virtual.

Thinkify is a privately held company, located in Morgan Hill, California.

We feel that partnerships should be healthy and that Engineering should be beautiful.

**Thinkify, LLC**
18450 Technology Drive, Suite E1
Morgan Hill, CA 95037

Phone: 408.782.7111
FAX: 408.782.2111
Web: www.thinkifyit.com

*Thinkify – Making things think.* (tm)

## Table of Contents

# A Quick RFID Introduction

## *Class 1 Generation 2 (Gen2)*

The RFID tags included in your reader kit conform to the UHF Class 1, Generation 2 (“Gen2”) standard maintained by EPCglobal (http://www.epcglobalinc.org/). EPCglobal is a division of UPC - the same standards organization that controls the barcode numbering system used on retail packaging. This standard (with minor changes) is also maintained by ISO under ISO-18000-6C.

Most Gen2 tags are *passive* RFID devices. That is, they do not require a battery and derive their power for operation from the RF field sent out by the reader. This allows them to be small, inexpensive, and operate virtually indefinitely.

Most Gen2 tags are also programmable devices. Users can put their own information into the tags. The amount of data that can be stored depends on the type of tag but hundreds of bits are typical. Data in the tag is organized into banks of memory that serve different functions under the protocol:

| | |
|---|---|
| Bank 0: | Reserved Memory: Kill and Access pass-codes |
| Bank 1: | EPC Memory: The unique tag identifier, typically 128 bits, and user-programmable. The Gen2 protocol is designed to extract this information quickly. |
| Bank 2: | TID Memory: A factory-programmed area that includes a serial number and fields that describe the tag's capabilities. |
| Bank 3: | User Memory: A programmable extended memory area for holding additional information that is not the EPC. Not all tags support User Memory. |

Gen2 tag memory can be locked, such that it cannot be changed without a pass-code. These locks can be reversible or permanent (permalocked).

Bank 0 is a special case for locking. Locking the other banks prevents them from being changed. If Bank 0 is reversibly locked it cannot be read without a pass-code. If it is permalocked it can never be read again. This secures the Kill and Access pass-codes from unauthorized users.

Finally, Gen2 tags can be rendered non-functional with a “Kill” command.  Tags that are killed become nonfunctional and cannot be recovered.

## *Concepts (Performing an Inventory)*

Being an RFID reader trying to read multiple tags using the Gen2 protocol is sort of like being a new teacher trying to take attendance in a kindergarten class... Sadly, the administration didn't give you an attendance list on the first day of class so you have to work it out for yourself.

| Kindergarten Teacher | RFID Reader |
|---|---|
| You have to get a list of everyone's name. | You have to get a list of all of the EPC codes from the tags. |
| Kids know their own names. | Tags have unique IDs in EPC memory they can report. |
| You can only hear one child at a time. | The reader can only process a signal from one tag at a time. |
| Kids want to all talk at once. | Multiple tags can respond at the same time. |

What both the teacher and the RFID reader need is an *anti-collision protocol* – a way to keep their respective kids/tags from talking at the same time.

Most teachers adopt an adult-talks-first protocol with a persistent state flag for whether a child has been inventoried. This flag is maintained in the child. Sometimes there's a bi-directional exchange with an ACK/NAK option. Hey! that's sounds a lot like Gen2.

| Teacher | Child | Gen2 Protocol |
|---|---|---|
| *"Ok everyone! Quiet down. It's time to take attendance."* | | Reader-talks-first. |
| *"Ok everyone! Hands up!"* | | Under Gen2 this is a *Select* command that establishes who's going to participate in the inventory – in this case, everyone. By putting their hands up, the child has set a flag that indicates he/she hasn't been inventoried, yet. |
| *"When I point to you, tell me your first name."* | | Granted this is a little contrived, but it's a little like the *Query* command in Gen2 that kicks off an inventory sequence. |
| The teacher randomly picks the first child, points to her and says, *"You!"* | *"Inga!"* | In Gen2, a tag responds to a *Query* with a random number that is used in the next command by the reader |
| *"Inga who?"* | *"Svenson!"* | This is like a Gen2 *ACK* (acknowledgment). It tells the tag/child that the reader/teacher heard their response and is now asking them for their data. |
| *"You!"* | *"Mikey!"* | At this point, Inga assumes that the teacher got her name, since |

| | | |
|---|---|---|
| (Pointing to the next child.) | | she's moved on to the next child. She puts her hand down and sets her state to "Inventoried". |
| *"Mikey who?"* | *"Jones!"* | |
| *"Pardon me."* | | If the reader doesn't understand the reply it can issue a *NAK* and try again. |
| *"Mikey who?"* | *"Jones!"* | |
| *"You!"* (On to the next child.) | | Mikey puts his hand down, too and sets his state to "Inventoried". |

And off they go…

When the teacher reaches the end of the round because she sees no more raised hands, she is done.

This is clearly contrived and an oversimplification of both the teacher's real-life protocol and Gen2, but it does captures some of the important features:

1. Inventories of the field need an anti-collision protocol to prevent multiple tags from talking at the same time.
2. An inventory can begin with one or more *Select* commands that establish who will participate in the inventory. (Teacher: "Ok, only the boys, put your hands up!")
3. The state of whether or not a tag has been inventoried is maintained in the tag.
4. In the process of singulating a tag, the reader gets a handle (the child's first name in this example) that it can use for additional operations with that tag (more on this below).

The analogy breaks down when you realize that unlike the teacher, the reader cannot see the inventoried state of the tags (hands in the air). If the teacher tried to take attendance of the class from behind a curtain, it would be a lot more difficult. Rather than pointing at a child and saying, "You!" to keep them from talking at once, a different protocol would be needed.

In Gen2, this is accomplished with the *Query* command. When the reader issues a *Query* command, it includes in the message a parameter called Q that the tags use to determine if they will respond immediately, or after some number of subsequent *QueryRep* commands. The number of *Query* or *QueryRep* commands the tag will wait to hear is determined randomly and can vary from 1 to $2^Q$.

By adjusting the Q parameter used in its *Query* commands, the reader can prevent multiple tags from responding simultaneously, most of the time. If there is a collision, the reader can adjust Q or just try again and let the tags roll a different random number. From your perspective as a user of the reader, these details don't usually matter (we adjust Q for you automatically) but they can be useful to know sometimes if you are trying to optimize performance.

## Concepts (Reading / Writing other data)

The Gen2 protocol is strongly oriented around the use case of rapidly reading the data in Bank 1 of Memory, the EPC. In supply chain applications there can be hundreds of tags moving past a read point and the reader needs to read them all as they go by.

Reading data in other banks of memory or programming tag memory builds off of the protocol we use for isolating tags and it extends it, allowing a "conversation" to take place with a tag that has been isolated, or "singulated".

To read User memory for example, the reader first isolates a tag with an inventory, and then uses the handle from the tag as part of a sequence of commands to get the User data. Programming is done in a similar manner.

In the Thinkify reader, we allow you to specify a number of "descriptors" that tell the reader what additional actions, if any, to take when it reads a tag. Descriptors can be used to Read additional memory areas, Write to memory, Lock and Unlock tag memory, and Kill tags.

This is a very powerful approach. By using Select commands (called "masking") as part of the inventory we can quickly specify that we are interested in performing an operation on just one, some, or all of the tags presented to the reader.

# Thinkify Reader Protocol Overview

Here we give an overview of the Thinkify Reader Protocol message structure and provide a high-level summary of the major command groups available to the user.

The Thinkify Reader Protocol (TRP) is a human-readable ASCII protocol that allows users and applications to set parameters for RF control, tag list acquisition, tag programming, and digital I/O behavior. TRP may also be used to acquire data from the reader and be notified of tag read events, I/O events, and reader status.

TRP is used across all Thinkify reader products and supported hardware interfaces including; RS232, USB, and Ethernet.

## *Command Structure*

The Thinkify Reader Protocol uses a Command-Response model. Communication is initiated by the Host, and the Reader responds with an acknowledgment or data.

Users may interact with the reader from a terminal program or their own software using the Thinkify APIs. All that is required is that they send strings to the device over an active connection, and terminate messages correctly. Replies are sent back, often on multiple lines, terminated by a “READY>” prompt.

### *Host Commands*

Host commands to the Reader are ASCII strings terminated with a Carriage Return. Line feed characters are ignored by the reader and may be sent without effect. The Reader does not echo commands back to the Host.

Valid command messages are composed of numeric characters in the range of 0-9 (0x30 - 0x39), ASCII characters in the range of a..Z (0x41 - 0x7A), and the carriage return character (0x0D).

The general format of a Host-to-Reader message is:

```
<COMMAND>[<SUBCOMMAND>[<PARAM1>][<...>][<PARAMn>]]<CR>
```

(here [ ] denotes an element that may be optional)

| | | |
|---|---|---|
| `<COMMAND>` | – | Typically a single character. |
| `<SUBCOMMAND>` | – | Typically a single character. |
| `<PARAMs>` | – | Vary in length and depend on the command being sent. There are no spaces between parameters, if multiple parameters are sent as |

|  |  |  |
|---|---|---|
|  |  | part of a message. |
| `<CR>` | – | The Carriage Return character (0x0D). |

Upon receipt of a carriage return, the Reader will attempt to parse the command message and, if it is properly formatted, execute the command.

## *Reader Replies*

The reply the Reader makes to Host commands are also ASCII strings. Replies may either be a single line or a multi-line reply, depending on the Command. Each line of a reply is terminated with a Carriage Return + Line Feed character pair, CRLF (0x0D,0x0A).

When the reader has finished sending all data back to the host in response to the command, it will end the sequence with a “READY>” prompt, indicating that it is prepared to process another message. Generally, after sending a Command, the Host should not send a new command until it sees the "READY>" message.

The general format of a Reader-to-Host message is:

```
[STARTMSG<CRLF>]
<Line1><CRLF>
<Line2><CRLF>
…
<Linen><CRLF>
[STOPMSG<CRLF>]
<CRLF>
READY>
```

(here [ ] denotes an element that may be optional)

|  |  |  |
|---|---|---|
| `[STARTMSG]` | – | Indicates the beginning of command processing. Not sent on every command, but is when inventories are performed. |
| `<Lines>` | – | Data sent back in response to the command. |
| `[STOPMSG]` | – | Indicates command processing is finished. Not sent on every command, but is when where inventories are performed. |
| `READY>` | – | Indicates that the reader is ready to accept another command. |

### *Special Case: Inventory Replies*

When the Reader performs a T or Tn command that is setup for infinite repeat, it streams line data until it sees a character from the host. It then terminates the message with the STOPMSG and READY> prompt.

## Examples

### 1. Set the General Purpose Output (GPO) Pin 1 to a High Level:

```
<COMMAND=”G”><SUBCOMMAND=”1”><PARAM1=”1”><TERM=0x0D>
```

The Host would send the string:

```
G11<CR>
```

The Reader would respond with:

```
GPOUT1=1<CRLF>
READY>
```

### 2. Read Tags using the “T” command:

```
<COMMAND=”T”>
```

Host:

```
T<CR>
```

Reader:

```
STARTINVENTORY<CRLF>
TAG=3000100000000000000000003560<CRLF>
TAG=3000100000000000000000003568<CRLF>
TAG=300010011002100310041007BBBB<CRLF>
TAG=3000100000000000000000003583<CRLF>
TAG=3000100000000000000000003556<CRLF>
TAG=3000100000000000000000003569<CRLF>
TAG=3000100000000000000000003557<CRLF>
TAG=3000BBAA99887766554433221100<CRLF>
TAG=3000100000000000000000003582<CRLF>
STOPINVENTORY 0x00EA EPCCOUNT=9<CRLF>
<CRLF>
READY>
```

### 3. Query the Inventory Parameter Settings:

```
<COMMAND=”I”>
```

Host:

```
I<CR>
```

Reader:

```
SESSION=1<CRLF>
TARGET=0<CRLF>
Q=3<CRLF>
OUTERLOOP=01<CRLF>
INNERLOOP=03<CRLF>
SELECTLOOP=1<CRLF>
SELTYPE=0
LOOPDELAY=0
<CRLF>
READY><CRLF>
```

### 4. Tn Command:

The Tn (T1, T2, ...T6) commands repeatedly perform inventories until interrupted by the Host. During this time the Reader streams tag data until a character is received from the Host. The reader then stops the Inventory sequence and terminates the reply.

Host:

```
T6<CR>
```

Reader:

```
STARTINVENTORY<CRLF>
TAG=3000100000000000000000003582 911750 07 8 9 Q E468<CRLF>
TAG=3000100000000000000000003557 911750 04 8 9 I E471<CRLF>
TAG=3000100000000000000000003583 911750 06 8 9 Q E47C<CRLF>
TAG=3000100000000000000000003557 911750 02 8 9 I E486<CRLF>
TAG=3000100000000000000000003557 911750 06 8 9 I E493<CRLF>
TAG=3000100000000000000000003568 911750 02 8 9 Q E49D<CRLF>
TAG=3000100000000000000000003557 911750 07 9 A I E4A9<CRLF>
TAG=3000BBAA99887766554433221100 911750 02 9 A Q E4B4<CRLF>
TAG=3000100000000000000000003556 911750 07 7 0 I E4C3<CRLF>
TAG=3000100000000000000000003557 911750 00 7 0 Q E4D3<CRLF>
TAG=3000100000000000000000003557 911750 05 7 0 Q E4DD<CRLF>
TAG=3000100000000000000000003569 911750 06 7 0 I E4ED<CRLF>
TAG=3000100000000000000000003583 911750 04 7 0 I E4F5<CRLF>
TAG=3000100000000000000000003560 911750 02 7 0 Q E4FD<CRLF>
TAG=3000100000000000000000003557 911750 00 7 0 Q E506<CRLF>
```

(Character, such as <SPACE> received from the Host)

```
 TAG=3000100000000000000000003569 911750 07 7 1 I E511<CRLF>
TAG=3000100000000000000000003557 911750 01 7 1 Q E51C<CRLF>
STOPINVENTORY 0x00C6 EPCCOUNT=17<CRLF>
<CRLF>
READY><CRLF>
```

## Command Groups

Commands are grouped into five major areas, and described in the following sections.

1. **Tag [I, K, M, T, X]** (for working with RFID tags)
2. **GPIO & Triggering [G]** (for interacting with the reader's GPIO port)
3. **Radio Control [P, R]** (for controlling the reader's radio subsystem)
4. **System [B, S, V]** (for firmware updates, version #, etc.)
5. **Engineering Test [A, C, D, F, L]** (advanced engineering functions used mostly for regulatory testing and by users wishing to develop custom OEM solutions.)

# Command Reference

## *Summary*

| Main Command | Description | Command Group |
|---|---|---|
| **A** | RX Amplifier Control | Engineering / Test |
| **B** | Enter Bootloader | System |
| **C** | Low-Level Chip Registers | Engineering / Test |
| **D** | Diagnostic Functions | Engineering / Test |
| **F** | RX Filter Control | Engineering / Test |
| **G** | GPIO Control | GPIO Control and Triggering |
| **I** | Inventory Control | Tag Commands |
| **K** | Kill / Access Data Descriptors | Tag Commands |
| **L** | Low-Level Tests | Engineering / Test |
| **M** | Tag Masking | Tag Commands |
| **P** | Protocol Air Interface | Radio Control |
| **R** | RF Control | Radio Control |
| **S** | Status Functions | System |
| **T** | Perform Tag Inventory | Tag Commands |
| **V** | Get Firmware Version (Read Only) | System |
| **X** | eXtra Read / Write Data Descriptors | Tag Commands |

## "A" – RX Amplifier Control

```
A[<SUBCMD>[<PARAMS>]]
```

The "A" command and sub-commands are used to set and get the parameters that control the characteristics of the amplifier in the base band receiver.

### Sub-Commands

| Sub Command | Description | Legal Values for SET |
|---|---|---|
| **A** | Report all RX Amplifier settings. | - |
| **AA** | 8 dB mixer attenuation control. 0=Off, 1=On. | 0..1 |
| **AG** | Gain adjustment:<br>**Value** **Gain**<br>0 0dB<br>1 -9dB<br>2 -6dB<br>3 -3dB<br>4 +3dB<br>5 +6dB<br>6 +9dB | 0..6 |
| **AH** | Hysteresis: 7 steps of 3dB each. | 0..7 |
| **AM** | 10 dB mixer amplification control. 0=Off, 1=On. | 0..1 |

### "A" Command Examples

#### Get All Settings

```
READY>a
GAIN=0
HYSTERESIS=0dB
MIXERBOOST=0
```

#### Set the Gain to -6dB

```
READY>ag2
GAIN=-6
```

#### Add 9dB of Hysteresis

```
READY>ah3
HYSTERESIS=9dB
```

#### Turn on Mixer Boost

```
READY>am1
MIXERBOOST=-8
```

## *"BOOTLOADER" – Enter Bootloader*

```
BOOTLOADER
```

Places the reader in a special mode where it is waiting to receive a firmware upgrade. In this state, the reader will not respond to normal commands and requires a power cycle to return to normal operation. See Appendix A for how to upload firmware using the Thinkify Upgrade Utility.

**Note:**
Entering bootloader mode un-enumerates the USB port in Windows. Reset into normal code re-enumerates port.

This can confuse terminal programs like Tera Term / Hyperterm. After executing the bootloader command disconnect terminal program. After resetting and re-enum then reconnect terminal program.

The host Bootloader program provided by Thinkify for firmware upgrades runs the USB interface with a HID windows class driver. (Normal operation is with a windows CDC class driver.)

### *“BOOTLOADER” Command Examples*

```
READY>bootloader
ENTERINGBOOT
```

The reader is now waiting for a firmware upgrade. At this point you may use the Thinkify Upgrade Utility to load new firmware. See Appendix A.

To reboot the reader, simply unplug it and plug it back in.

## "C" – Low-Level Chip Control Registers

```
C[<SUBCMD>[<PARAMS>]] - See table, below.

C<ADDR><VAL>          - Sets a register.
                        ADDR may be one or two nibbles.
                        VAL may be 2 or 6 nibbles.
```

The “C” command and sub-commands are used to set and get the low-level control registers in the AM3392 chip. (This is an engineering command.)

### Sub-Commands

| Sub Command | Description | Legal Values for SET |
|---|---|---|
| **c** | Report all control registers. | -c |

| **Register** | **Description** |
|---|---|
| 0x00 | Status control (byte) |
| 0x01 | Protocol control (byte) |
| 0x02 | TX option (byte) |
| 0x03 | RX option (byte) |
| 0x04 | TRcal Low reg (byte) |
| 0x05 | TRCal Hi reg (byte) |
| 0x06 | TX Delay (byte) |
| 0x07 | RX No Resp Wait (byte) |
| 0x08 | RX Wait (T1) (byte) |
| 0x09 | RX Filt Reg (byte) |
| 0x0A | RX Spec2 (byte) |
| 0x0B | Regulator and RF control (byte) |
| 0x0C | - |
| 0x0D | IRQ Mask (byte) |
| 0x0E | |
| 0x0F | |
| 0x10 | |
| 0x11 | Test Select Reg (byte) |
| 0x12 | Test Setting reg (word) |

| Register | Description |
|---|---|
| 0x13 | |
| 0x14 | CLSYS ANAOUT (word) |
| 0x15 | MOD control (word) |
| 0x16 | PLL main control (word) |
| 0x17 | PLL aux control (word) |
| 0x18 | DAC reg (byte) |
| 0x19 | |
| 0x1A | RXLen1 (byte) |
| 0x1B | RXLen2 (byte) |
| 0x1C | |
| 0x1D | TXLEN1 (byte) |
| 0x1E | TXLEN2 (byte) |

| Sub Command | Description | Legal Values for SET |
|---|---|---|
| **CS** | Report all shadow registers | - |
| **CR** | Resets all registers to program default. | - |

## *"C" Command Examples*

### *Get All Control Register Values*

```
READY>c
ICREGISTER00=00
ICREGISTER01=00
ICREGISTER02=00
ICREGISTER03=00
ICREGISTER04=00
…
ICREGISTER3C=00
ICREGISTER3D=00
ICREGISTER3E=00
```

### *Get All Shadow Register Values*

```
READY>cs
ICSHADOW=00=00
ICSHADOW=01=00
ICSHADOW=02=32
ICSHADOW=03=6B
ICSHADOW=04=05
…
ICSHADOW=3C=01
ICSHADOW=3D=00
ICSHADOW=3E=00
```

## "F" – RX Filter Control

```
F[<SUBCMD>[<PARAMS>]]
```

The “F” command and sub-commands are used to control the RX baseband filter. These commands may be used in troubleshooting and regulatory testing. (This is an Engineering test function.)

### Sub-Commands

| Sub Command | Description | Legal Values for SET |
|---|---|---|
| **F** | Report current filter settings. | - |
| **FL** | Low Pass Value | 0..7 |
| **FH** | Hi Pass value | 0..7 |
| **FB** | Bypass Filters<br>Bit 0 = 40 KHz<br>Bit 1 = 160 KHz | 0..3 |
| **FS** | AC Speedup. 1=On, 0=Off. | 0..1 |

### “F” Command Examples

#### Get Current Filter Settings

```
READY>f
LOWPASS=7
HIGHPASS=7
BYPASS160=0
BYPASS40=0
```

#### Set High Pass to 3

```
READY>FH3
HIGHPASS=3
```

#### Set Low Pass to 5

```
READY>FL5
LOWPASS=5
```

#### Turn On 40 and 160kHz Bypass Filters

```
READY>fb3
BYPASS160=1
BYPASS40=1
```

## "G" – GPIO Settings

```
<G>[<SUBCMD>[<PARAMS>]]
```

The G command and sub-commands are used to control the GPIO port. These may be used to set/retrieve GPIO pin settings or to set the reader up for triggered reading.

Using the GT command, the reader may be configured to read tags in any of the supported inventory modes for either a fixed time after an edge transition or while a pin is held in a particular state.

### Sub-Commands

| Sub Command | Description | Legal Values for SET |
|---|---|---|
| **G** | Reports current state of input and output lines. | - |
| **G0** | Write Output Port 0 (no Get) | 0..1 |
| **G1** | Write Output Port 1 (no Get) | 0..1 |
| **GT** | Triggering setup for Autonomous Reading<br>`   GT<port><active>[<type><action><time>]`<br>`     <port>    0/1 trigger on INPUT0/1`<br>`     <action>  0/1 = disable/enable trigger`<br>`   if <action>=1, then include:`<br>`     <type>    0=posEdge,  1=negEdge,`<br>`               2=posLevel, 3=negLevel`<br>`     <action>  0=T3, 1=T4, 2=T5, 3=T6, 4=T`<br>`     <time>    if pos/negEdge only; range is 0x01 to`<br>`               0xFF in .1sec units (.1 to 25.5 sec)` | See Description |

### "G" Command Examples

#### Get Current I/O States

```
READY>g
GPIN0=1
GPIN1=1
GPOUT0=0
GPOUT1=0
```

#### Turn Output Port 0 On

```
READY>g01
GPOUT0=1
```

#### Get Trigger Settings

```
READY>gt
GPTRIGINVENTORY=DISABLED
PORT=0
INVENTORYTYPE=6
TRIGTYPE=POSEDGE
TIMER=0
```

#### Configure Edge Trigger w/Timer

```
READY>gt1150
GPTRIGINVENTORY=ENABLED
PORT=1
INVENTORYTYPE=5
TRIGTYPE=POSEDGE
TIMER=0
```

#### Turn the Trigger Off

```
READY>gt00
TRIGGERTYPE=DISABLED
```

## *"I" – Inventory Control*

```
I[<SUBCMD>[<PARAMS>]]
```

The I command and sub-commands are used to set and get the parameters that control the flow of the Gen2 anti-collision algorithm. Modifications to the default parameters may be helpful in cases where there are a large number of tags in the field or when it is desirable to increase the number of redundant reads for a given tag.

### *Sub-Commands*

| Sub Command | Description | Legal Values for SET |
|---|---|---|
| **I** | Display all of the Inventory Control settings. | - |
| **IB** | When performing a write operation as part of an inventory sequence, a read operation is usually performed before the write.<br><br>Issue:<br>IB0 to send the read before the write.<br>IB1 to block sending of the read before the write<br><br>In Blockwrite operations, you may choose to issue a ReqRN command before the Blockwrite. (Needed for NXP G2iL+ block write)<br><br>Issue:<br>IB2 to turn OFF send REQRN before the BLOCKWRITE.<br>IB3 to turn ON send REQRN before the BLOCKWRITE | - |
| **IN** | **NOTAG reporting**<br>• This will report a message if the whole round (all slots) did not find any tag.<br>• Useful for evaluating performance, for example: looking for frequencies that do not read well…or comparing how well setup values for the receiver or the protocol are working. | 0 or 1 |
| **IR** | **Reset** inventory parameters to values. | - |
| **II** | **Inner Loop Count**<br>Each INNERLOOP runs a tag acquisition STATEMACHINE. | 0..FF |
| **IL** | **Gen2 SEL Flag**<br>Value used in QUERY for the SEL field. See G2 spec. Usually set to 0. | 0..3 |
| **IO** | **Oneshot Mode**<br>inventory will terminate after first tag is seen | 0 or 1 |
| **IP** | **Outer Loop Pause Time**<br>Time in msec to delay after each outer loop before starting another | 0..99999 (Decimal) |

| Sub Command | Description | Legal Values for SET |
|---|---|---|
| | inventory cycle. (Allows duty cycling for low power applications.) This is a DECIMAL quantity ranging from 0 to 99999 msec. | |
| **IQ** | **Gen2 Q Parameter**<br>The Q used in the QUERY that starts the round | 0..8 |
| **IS** | **Gen2 Session**<br>The session (0 to 3) that will be used for the entire inventory run. | 0..3 |
| **IT** | **Inventory Target**<br>Defines whether the QUERY that initiate round is looking for tags in the A or B state | 0..1 |
| **IW** | **Select Count**<br>Number of times SELECT function is executed - each execution sends every MASK that is enabled | 0..F |
| **IX** | Append **XEPCDATA** to T output | 0..1 |

## *"I" Command Examples*

### *Get All Inventory Control Parameters*

```
READY>i
SESSION=1
TARGET=0
Q=3
INNERLOOP=01
SELECTLOOP=1
SELTYPE=0
LOOPDELAY=0
```

### *Get Just the Q Value*

```
READY>iq


Q=3
```

### *Set Q to 5*

```
READY>iq5
Q=5
```

### *Set InnerLoops to 4*

```
READY>ii4
INNERLOOP=04
```

## *"K" – Kill, Lock, Access Descriptors*

```
K[<SUBCMD>[<PARAMS>]]
```

The K family of commands are used to control lock kill and access command behavior. The K commands allow the user to get/set passwords used in kill, lock and access operation and specify lock type for the lock commands.

The Kill, Lock and Access commands are described in detail in the EPC Global C1G2 specification: uhf c1g2_standard- version 1.2.0.pdf

### *Locking*

Locking is one of the more complex activities performed under the Gen2 protocol. As mentioned above, tag memory is divided into different regions or “Banks”. Tag memory may be “locked” where it can only be changed using an access password, or “perma-locked” where it cannot ever be changed again. (Other options, like “perma-unlock” are also available).

EPC, TID and User memory are *always readable* under standard Gen2 – *even when those regions are locked*. In contrast, Reserved memory, where the Kill and Access passwords are stored, can only be read with the correct access password after that section has been locked.

To lock or unlock a tag, first one must have a tag programmed with a non-zero access password written into the correct region of Reserved memory. Then, a Lock command may be issued with a data field representing which region(s) of memory to lock and what type of lock to use (regular or “perma” lock). The data field is a mask, where bits represent memory locations and lock types. (more below).

The following options are available for locking:

- **Read Unlocked** – there is unrestricted access to read from this memory
- **Read Locked** – the memory cannot be accessed for reading without using a password
- **Write Locked** – the memory cannot be accessed for writing without using a password
- **Read Permanently Unlocked** – there is unrestricted access to read from this memory and this memory can never be locked
- **Write Permanently Unlocked** – this memory cannot be accessed for writing without a password and this memory can never be locked

There are five sections of memory that can be each individually locked:

1. **Kill password**
2. **Access password**
3. **EPC memory**
4. **TID memory**
5. **User memory**

The Gen2 protocol specification referenced above describes the data fields associated with the Lock command. The data is a 20 bit number consisting of a 10 bit Mask field and an associated 10 bit action field. In the Insight reader, we use this number with the KL command lock descriptor to control the Locking behavior. The meaning of each bit is described in the table below.

Lock Data Fields, Mask Fields (Bits 10-19)

| Kill Password | | Access Password | | EPC Memory | | TID Memory | | User Memory | |
|---|---|---|---|---|---|---|---|---|---|
| 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |
| Skip (0) Write(1) | Skip (0) Write(1) | Skip (0) Write(1) | Skip (0) Write(1) | Skip (0) Write(1) | Skip (0) Write(1) | Skip (0) Write(1) | Skip (0) Write(1) | Skip (0) Write(1) | Skip (0) Write(1) |

Lock Data Fields, Action Fields (Bits 0-9)

| Kill Password | | Access Password | | EPC Memory | | TID Memory | | User Memory | |
|---|---|---|---|---|---|---|---|---|---|
| 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Password Read and Write | Perma Locked | Password Read and Write | Perma Locked | Password Write | Perma Locked | Password Write | Perma Locked | Password Write | Perma Locked |

We will use this table in an extended example for locking below.

### *Sub-Commands*

| Sub Command | Description | Legal Values for SET |
|---|---|---|
| **KA** | Get or set the Access password.<br>`KA reports the Access password,`<br>`KA<ACCESSPASSWORD> sets the Access password.` | 32 Bits from 8 Nibbles |

| | | |
|---|---|---|
| **KAR** | Resets Access password to the default. | - |
| **KL** | Get or set the Lock Descriptor.<br>Options:<br>`KL - Report Lock Descriptor`<br>`KL<active 1:0> - (De)Activate Lock descriptor`<br>`KL<active 1:0><LOCKBITS (20 bits in 5 ASCII HEX nibbles)> (De)Activate Lock descriptor and Set LOCK value` | See Description |
| **KK** | Controls KILL descriptor<br>`KK  report KILL descriptor`<br>`KK<active 1:0> activate or de-activate the KILL descriptor`<br>`KK<active 1:0><KILLPASSWORD (16 bit 4 ASCII HEX nibbles)> = activate or de-activate the KILL descriptor and setup KILL password value` | See Description |

## *"K" Command Examples*

### *Get Access Password*

```
READY>ka
ACCESSPASSWORD=00000000
```

### *Get Lock Descriptor*

```
READY>kl
ACTIVE=0
LOCKBITS=00000
```

### *Get Kill Descriptor*

```
READY>kk
ACTIVE=0
KILLPASSWORD=00000000
```

### *Set the Lock Active*

```
READY>kl1
ACTIVE=1
LOCKBITS=00000
```

### *Extended Example*

In this example we lock the kill and access passwords and note we can't see them without going into secure state.

Let's start with an unlocked tag with a kill / access password already programmed into reserved memory (see the XW commands).

Set up read descriptors to read reserved, epc, tid and user.

```
xr010400 (reserved)
xr111800 (epc)
xr212400 (tid)
xr313200 (user)
```

Perform an inventory and look at the results

```
t61

STARTINVENTORY

TAG=3000E2001AC1909F6580000EED95 902250 00 0 E Q 0AD5
XRD0=1111111122222222
XRD1=5F7D3000E2001AC1909F6580000EED95
XRD2=E2003414011F0100
XRD3=00000000
STOPINVENTORY 0x004D EPCCOUNT=1
```

you can see the access password is 22222222

setup the lock descriptor to lock (not perma) the kill and access passwords. This is the 20 bit number described in the table above.

```
10 Mask bits and 10 Action bits...

1010 0000 0010 1000 0000
   A    0    2    8    0
```

Issue lock descriptor with this data

```
kl1a0280

ACTIVE=1
LOCKBITS=A0280

set the access password
ka22222222

ACCESSPASSWORD=22222222
```

Now do an inventory w/access to lock.

```
t61

STARTINVENTORY
TAG=3000E2001AC1909F6580000EED95 902250 01 0 E Q E573
ACCESS=SUCCESS
XRD0=1111111122222222
XRD1=5F7D3000E2001AC1909F6580000EED95
XRD2=E2003414011F0100
XRD3=00000000
LOCK=SUCCESS
STOPINVENTORY 0x0001 0x006E
```

Turn off access password

```
kar

ACCESSPASSWORD=00000000
```

Turn off locking

```
kl0

ACTIVE=0
LOCKBITS=A0280
```

Try to read without access:

```
t6

STARTINVENTORY
TAG=3000E2001AC1909F6580000EED95 902250 02 0 E Q 4A35
XRD0=TAG ERRORCODE 04
XRD1=5F7D3000E2001AC1909F6580000EED95
XRD2=E2003414011F0100
XRD3=00000000
STOPINVENTORY 0x004A EPCCOUNT=1
```

You can't see the access password or kill password!

Use the right access password and go into secure state:

```
ka22222222

ACCESSPASSWORD=22222222

t61

STARTINVENTORY
TAG=3000E2001AC1909F6580000EED95 902250 03 0 E Q 6812
ACCESS=SUCCESS
XRD0=1111111122222222
XRD1=5F7D3000E2001AC1909F6580000EED95
XRD2=E2003414011F0100
XRD3=00000000
STOPINVENTORY 0x0060 EPCCOUNT=1
```

With the right access password, we can now read the Locked Reserved memory.

## *“M" – MASK / SELECT control*

```
M[<SUBCMD>[<PARAMS>]]
```

As mentioned in the introductory sections, an inventory may begin with the issuance of one or more Gen2 *SELECT* commands to determine which tags participate in the inventory round.

When the Select loop runs (see the IW command) each pass through the loop can issue up to four (4) independent Select commands. The parameters associated with these Select commands are stored in the reader's list of Masks.

When the Select is sent, the ACTIVE flag of each of the four masks is examined in order from 0 to 3. If ACTIVE == 1, the MASK is used as part of the Select command.

By default, MASK0 is active (ACTIVE FLAG 1) with an ACTION of 0 (all tags to A state) and a LEN of 0×00 (this means “select all tags”). See the G2 specification, table 6.19, for the eight different possible ACTIONS.
http://www.epcglobalinc.org/standards/uhfc1g2/uhfc1g2_1_2_0-standard-20080511.pdf

By default, MASK1, MASK2, MASK3 are set to INACTIVE (ACTIVE FLAG == 0).

### *Sub-Commands*

| Sub Command | Description |
|---|---|
| **M** | Report the mask descriptors for all four masks. |
| **M<#>** | Report the mask descriptor for just mask <#>, where <#>=0..3. |
| **M<#><...>** | Set the descriptor for mask <#>, where <#>=0..3. When setting the Mask, the format is:<br>`M<#><PARAMS>`<br><br>Where <PARAMS> includes the following :<br>`<ACTIVE><TTYPE><ACTION><MEMBANK><LEN><EBVBANK><MASKBYTES>`<br><br>`<ACTIVE>`<br>0=inactive, 1=active<br><br>`<TTYPE>`<br>0=use the current Session (see the IS command)<br>1=use SL 100 flag<br><br>`<ACTION>`<br>0-7, usually use 0. See the table, below, for a summary of the eight actions, or the EPCglobal G2 Spec, table 6.20, for more details. |

| Sub Command | Description |
|---|---|
| | **Action** / **Matching** / **Non-Matching**: see table below. `<MEMBANK>` 0=Access & Kill Passwords, 1=EPC, 2=TID, 3=USER. `<LEN>` A byte indicating the number of *bits* in the mask. `<EBVBANK>` 1-4 bytes, this is a *bit* pointer - see annexA G2 spec about EBV pointers. `<MASKBYTES>` 0 to 32 bytes, representing the mask data. There must be enough bytes to meet the indicated <LEN>. All bits are *left* justified (i.e. MSB of BYTE0 is the first bit of mask, MSB of BYTE1 is 8th bit of mask etc.) |
| **MR** | Set Mask parameters to default values. |

| Action | Matching | Non-Matching |
|---|---|---|
| **0** | assert **SL** or **inventoried** → *A* | deassert **SL** or **inventoried** → *B* |
| **1** | assert **SL** or **inventoried** → *A* | do nothing |
| **2** | do nothing | deassert **SL** or **inventoried** → *B* |
| **3** | negate **SL** or (*A* → *B*, *B* → *A*) | do nothing |
| **4** | deassert **SL** or **inventoried** → *B* | assert **SL** or **inventoried** → *A* |
| **5** | deassert **SL** or **inventoried** → *B* | do nothing |
| **6** | do nothing | assert **SL** or **inventoried** → *A* |
| **7** | do nothing | negate **SL** or (*A* → *B*, *B* → *A*) |

`<MEMBANK>`
0=Access & Kill Passwords, 1=EPC, 2=TID, 3=USER

`<LEN>`
A byte indicating the number of *bits* in the mask.

`<EBVBANK>`
1-4 bytes, this is a *bit* pointer - see annexA G2 spec about EBV pointers.

`<MASKBYTES>`
0 to 32 bytes, representing the mask data. There must be enough bytes to meet the indicated <LEN>. All bits are *left* justified (i.e. MSB of BYTE0 is the first bit of mask, MSB of BYTE1 is 8th bit of mask etc.)

## *“M” Command Examples*

This can be tricky so let's work it out with an example:
`Tag=3000BBAA99887766554433221100`

With this ID we have an EPC bank with data in the following hex bit positions:

| **EPC Data** | xxxx (CRC) | 3000 (PC) | BBAA | 9988 | 7766 | 5544 | 3322 | 1100 |
|---|---|---|---|---|---|---|---|---|
| **Bit Position (Hex)** | 0x00 | 0x10 | 0x20 | 0x30 | 0x40 | 0x50 | 0x60 | 0x70 |

Notice how there are CRC and PC words (“3000”) before the actual EPC starts (“BBAA”)? Say we want to mask on the first part of the EPC code of this tag, "BBAA", we would have to use a pointer of 0x20 into the EPC bank.

Now recall the structure or the Mask command and its parameters:

```
M +
MASKNUM +
ACTIVE +
TTYPE +
ACTION +
MEMBANK +
LEN(1 byte)+
EBV(1 byte MIN) +
DATA
```

To set mask #0 to look for “BAAA” in the right position we'd say:

```
M + 0(mask) + 1(enable) + 0(ttype) + 0(action)+ 1(EPC bank) + 10(16 bits) +
20(pointer) + BBAA(data)
```

Our mask command would be:
**`M010011020BAAA`**

We try this out below...

### *Get Mask #0*

```
READY>m0
MASK=0
ACTIVE=1
TARGET=1
ACTION=0
BANK=1
PNTR=00
LEN=00
BITS=ZEROLENGTH
```

### *Get All Masks*

```
READY>m
MASK=0
ACTIVE=1
TARGET=1
ACTION=0
BANK=1
PNTR=00
LEN=00
BITS=ZEROLENGTH

MASK=1
ACTIVE=0
TARGET=1
…

MASK=3
ACTIVE=0
TARGET=1
ACTION=0
BANK=1
PNTR=00
LEN=00
BITS=ZEROLENGTH
```

### *Set Mask #0*

```
// Look for some tags...
READY>t5
STARTINVENTORY
TAG=30001000000000000000000003557
TAG=30001000000000000000000003582
TAG=3000BBAA99887766554433221100
TAG=30001000000000000000000003560
STOPINVENTORY 0x01C8 EPCCOUNT=14

// Report only our favorite tag
READY>m010011020bbaa
MASK=0
ACTIVE=1
TARGET=1
ACTION=0
BANK=1
PNTR=20
LEN=10
BITS=BBAA

READY>t5
STARTINVENTORY
TAG=3000BBAA99887766554433221100
TAG=3000BBAA99887766554433221100
TAG=3000BBAA99887766554433221100
...
STOPINVENTORY 0x028C EPCCOUNT=10
```

## *“NM" – Macro Control*

The Macro is a place to store and save reader settings that are preserved even after a device power off/on or BRS (reset command).

| Sub Command | Description |
|---|---|
| **NMA** | Macro Append - add to the end of the macro list without erasing the rest (so you can add settings without typing the whole thing over) |
| **NMC** | Erase the macro list |
| **NMP** | Play the macro list (will be done automatically on reset). |
| **NMW** | Erase current macro list first, type END to stop the writing operation. |
| **NMR** | Read the current macro list |

### *Macro Append*

```
READY>nmaag5
APPEND SUCCESS

READY>nmr
START USER MACROREAD
AG5
END
USER MACRO END
```

### *Macro Clear*

```
READY>nmc
USER MACROERASE

READY>nmr
START USER MACROREAD
END
USER MACRO END
```

### *Macro Write*

```
READY>nmw
USER MACROERASE
USER MACRORECORD START
ag5
p022
end
USER MACRO WRITE COMPLETE
```

## *"P" – PROTOCOL control (Gen2 Air protocol)*

```
P[<TARI><MODE><LF>]
      <TARI> = 0..2
      <MODE> = 0..3
      <LF>   = 0..4
```

The reader supports a number of different data rates and modulation modes for communicating with Gen2 RFID tags. This functionality is controlled by the P command.

Read performance is closely tied to how the various modulation, tag signaling and data rate parameters interact in a particular use case. Changes away from recommended settings should be done only after sufficient testing demonstrates an improvement. The best settings are often a compromise between read speed and read reliability. In some cases it may be beneficial to change this setting to improve performance in multi-reader environments.

Each of the <TARI>, <MODE>, and <LF> parameters can take on a simple integer value, corresponding to one of the available settings for that parameter, as outlined in the following table.

### *Available Parameters*

| Value | TARI | Modulation Mode (MODE) | Link Frequency (LF) |
|---|---|---|---|
| **0** | 6.25 uSec | FM0 | 40KHz |
| **1** | 12.5 uSec | M2 | 160 kHz |
| **2** | 25 uSec | M4 | 256 kHz |
| **3** | - | M8 | 320 kHz |
| **4** | - | - | 640 kHz |

### *Recommended Settings*

The default Protocol Control values from start up or a (PR – reset) command correspond to a setting of “231”, or a 25uSec TARI, M8, and a 160kHz LF. This is close to the DRM (Dense Reader Mode) requirement defined be EPCglobal, and works well in most situations.

Normal Operation : P222

Dense Reader Environment: P231

High Speed Reads of Small Numbers of Tags: P112

## *"P" Command Examples*

### *Read the current settings*

```
READY>p
TARI=12.5
M=M8
LF=250
```

### *Set 25 uS TARI, Miller 4, 250 kHz LF*

```
READY>p222
TARI=25.0
M=M4
LF=250
```

## "R" – RF Control

```
R[<SUBCMD>[<PARAMS>]]
```

The R command and sub-commands are used to monitor and control radio functions for power and RF frequency. -These commands are used during regulatory testing or under FCC Part 90, licensed operation of the device they are not to be changed outside of the specified limits except by qualified installers.

### Sub-Commands

| Sub Command | Description | Legal Values for SET |
|---|---|---|
| **RA** | RF transmitter attenuation control. A non-linear function that controls output power pf the RF transmitter (see below). | Settable Range is RA0 to RA19.<br><br>Legal values for unlicensed operation are values higher than the factory default setting. |
| **RO** | Control status of RF carrier (for test only).<br>RO1 = OFF<br>RO2 = IDLE<br>RO3 = ON | Do not Change.<br>Engineering test function. |
| **RF** | Get/Set the current RF frequency (for test only).<br>RFXXXXX (Five Decimal Numbers) | Do not Change.<br>Engineering test function. |
| **RH** | Get/Set the current hop dwell time (for test use only). | Do not Change.<br>Engineering test function. |

### "R" Command Examples

#### Get the Current Attenuation

```
READY>ra
ATTENUATION=10
```

#### Change the Attenuation

```
READY>ra12
ATTENUATION=12
```

#### Get the Current Frequency

```
// Watch it hop!
READY>rf
HOPPING
```

## ***The RA setting***

The RA setting controls power output. Higher values yield lower output powers. The command RA for output can be set from 8-19 on the TR-100.  Higher RA setting, for example RA19, means lower output power.  Lower RA setting, for example RA8, means higher output power.

In normal, unlicensed operation, the RA value should not be set below its factory default value. The default/minimum value is reported with the “RA” command after a reader start up.

As Thinkify continues to improve noise and efficiency in the TR-100, exact measurements of output power are somewhat open to small changes as we work on the reader.  **However, the reader will output about 17 dBm at the setting RA10**.

## *"RP" and "RM" – Multiplexing (Antenna Control)*

### *Sub-Commands*

| **Sub Command** | **Description** | **Legal Values for SET** |
|---|---|---|
| **RP<E or D>** | Show Antenna numbers in the inventory output | RPE to enable and<br>RPD to disable |
| **RP<Antenna number>** | Select the antenna to read from. Only use that antenna until a new antenna is selected. Using the RP command tells the reader to read from only one antenna. Note: using RP will automatically force MUXMODE to 0 | Antenna number can be set to 1-4 |
| **RMM<number of Antennas>** | Determine how many antennas take part in the mux sequence (For example: RMM2 would only use 2 antennas to sequence over) | 1-4 |
| **RM1** or **RM1<NUMINV>** | Sets Mux Mode 1<br>• Mode 1 is a simple ROUNDROBIN - a full set of inventory operations will be run on each antenna sequentially<br>• You can specify the number of times the full inventory is run with the command RM1<NUMINV(1 to F) where the inventory is run NUMINV+1<br>• RM1 sets NUMINV to 0<br>• This mode is useful if you have multiple antennas that are scanning for tags in uncoordinated areas where a given tag is probably only going to be seen by one of the antennas | n/a |
| **RM2** or **RM2<NUMINV>** | Sets Mux Mode 2<br>• Mode 2 is also a ROUNDROBIN - BUT one of the inventory operations (SELECT) is ONLY set on antenna 0<br>• For example, the antenna sequence is: 1(Do SELECT), 2, 3, 4, 1(Do SELECT), 2, 3, 4…<br>• This is useful where multiple antennas are scanning the same area for tags | n/a |
| **RM3** or **RM3<NUMINV>** | Sets Mux Mode 3<br>• Mode 3 is a ROUNDROBIN - but the antenna used for the select is incremented through the antenna sequence.<br>• For example, the antenna sequence is (Where S is SELECT): 1S, 2, 3, 4, 2S, 3, 4, 1, 3S, 4, 1, 2, 4S, 1, 2, 3 | n/a |

| | | |
|---|---|---|
| **RM4** or **RM4<MUXMR (2 nibbles) units are 10ms + fixed 25ms><PREEMPT (0 or 1)>** | Sets Mux Mode 4<br>• Mode 4 is a TIMED ROUNDROBIN - each antenna is used for a timed interval regardless<br>• This mode is mostly intended for RAIL operation, but could be used for GEN2 if desired (maybe with sub-optimum results)<br>• The amount of time spent on each antenna is set in the command | n/a |
| **RM5**<br>or<br>**RM5<ANT#1><ANT#2><ANT#3><ANT#4>**<br>or<br>**RM5<#1><#2><#3><#4><INV COUNT>** | Sets Mux Mode 5<br>• Mode 5 is used to select specific antennas to sequence over<br>• For example: rm50101 would sequence over ONLY antennas 2 and 4.<br>• INVCOUNT (INVCOUNT+1 passes through the state machine on each antenna) | n/a |

## *"R" Command Examples*

### *Append Antenna Numbers at the end of each Inventory line*

```
READY>rpe
ANTMUX=0 CURANT=1 COMMANT=1
```

### *Read on a single antenna*

```
READY>rp2
ANTMUX=0 CURANT=2 COMMANT=1
```

### *Set the number of antennas to sequence over*

```
READY>rmm2
MUXMAXANT=2
```

### *Set Multiplexing Mode*

```
READY>rm1
MUXMODE=1 INVCOUNT=0 MAXANT=4 MUXTMR=0 PREEMPT=0
```

### *Set Multiplexing Mode 5 (Select Specific Antennas)*

```
// Multiplex/Read with only antenna 1 and 4
READY>rm51001
MUXMODE5ARRAY=1001 INVCOUNT=0
```

## ***"T" – Initiate INVENTORY***

```
T
T<MODE>[<LOOPCOUNT>]
```

Attempt to read tags using the current settings.

### ***The "T" command***

The T command performs a full dual-nested loop sequence of: SELECT / QUERY / ACK / REQRN / ACK / XREAD / XWRITE, reporting tags as they are found, performing XDATA operations, and attempting to force found tags into the opposite A/B state. All aspects of this command are controlled by the reader's global inventory control parameters (see the "I" command), and the X data descriptor parameters (see the "X" command).

The parameters of the SELECT sequence sent in each OUTERLOOP are fully controllable through the mask commands (see the "M" command). Inclusion, exclusion, choice of A→B, B→A, etc. are all under user control.

The global parameters OUTERLOOP, INNERLOOP, SELECTLOOP, and Q can be over-ridden at the command line entry of the command, all other parameters are set globally through the I and X series commands. The T command will start at the requested Q value, but it will adjust Q depending on whether there are not enough tag responses (Q will be adjusted down) or too many response collisions (Q will be adjusted up).

If an OUTERLOOP value is set to 0xFF, then the T command will loop constantly until a character is received on the interface port. The same thing will occur on a T(n) with a loop value of 0xFF (equivalent to no loop value given).

The ISO-18000-6-C (Gen2) protocol specifies a set of low-level commands that can be used to read and write RFID tags. In practice, much of the detail surrounding how this is done is not important to the end user of an RFID system – you just care if the reader reports all the tags and that the data you want to write to them gets written correctly.

That said, some knowledge of what's going on can be used to optimize a system to improve read performance, programming reliability and efficiency. What you want to optimize depends on what you are trying to do with the RFID tags.

In some cases, you want to read a small number of tags very quickly and get lots of repeated reads of the same tag. An example of this might be an application where you are using an RFID tag on a runner to determine when he/she crosses the finish line of a race. The extra reads here are useful for determining the best "crossing time" for the runner.

In another case, you care less about the number of *redundant* reads and more about the number of *unique* reads you get. An example might be a tool tracking application where you are trying to read all the tagged items within a cabinet and don't want to miss any tags.

To handle these and other cases, you can issue a T command in conjunction with the M, I and X commands to fine-tune what is being reported from the tag field and how the reader interacts with the tag population it sees.

The output of the T command is formatted like this:

```
STARTINVENTORY
TAG=<epc1>
TAG=<epc2>
…
TAG=<epcN>
STOPINVENTORY 0x<Duration> EPCCOUNT=<N>
```

Each tag's EPC (including the PC word that precedes it) is listed on its own line, with “Tag=” in front of it. The entire list of tags is surrounded by “STARTINVENTORY” and “STOPINVENTORY 0x<Duration> EPCCOUNT=<N>”, where N is the number of tag acquisitions made (not unique tags), and Duration is how long the inventory took in milliseconds (e.g. 0x0200 = 512 msec = 0.512 sec).

You can also have the reader append XEPCDATA to each tag entry in the output of the T command. This XEPCDATA includes the following inventory- and protocol-related values at the instant when tag was acquired:

```
<FREQ> <OUTERLOOP> <INNERLOOP> <ROUND> <SLOTCOUNT> <Q> <TIMESTAMP>
```

XEPCDATA in the T output is enabled with the command:

```
IX1
```

If no tags are found in a T or T(n) command, a NOTAG message will be sent. In a T command, this means at every exit from the outer loop. In a T(n) command, this means when all slots for the current Q have been tried.

## *The “T<n>” commands:*

In addition to the basic T command, tags may also be acquired using the T<n> series of sub-commands. In these commands a minimal series of air protocol commands are issued to acquire the tag data, and the tags are not removed from the round with an A/B transition, so in general these commands are only useful when the tag population is small.

In each of the T<n> commands the number of slots tried will be determined by the global Q value (see the “IQ” command). The Masks sent in the commands that include a SELECT will be determined by the values in the global Mask structure array (see the “M” command). Any XDATA processing events will be determined by the values in the XDATADESCRIPTOR array (see the “X” command).

The odd-numbered T<n> sub-commands all report just the tag's EPC. The even-numbered

T<n> sub-commands perform the same inventory action as the odd-numbered sub-commands that precede them, except more information is provided in the tag report besides just “Tag=<epc>”:

```
        TAG=<epc> <freq> <slot> <Imag> <Qmag> <I/Qdecoded> <timestamp>
```

In all of the T<n> commands, sending the command alone causes the command to execute repeatedly, until a character is received over the interface port. If the T<n> command is followed by an optional one-byte <LOOPCOUNT> parameter, the command executes in a loop the number of times specified by <LOOPCOUNT>. Note that  providing a <LOOPCOUNT> value of 0xFF is the same as providing no value - a continuous loop occurs until a character is received on the interface port.

| Sub Command | Description | Special Features |
|---|---|---|
| **T1** | T1[<LOOPCOUNT>]<br>Sends a QUERY/QUERYREP/ACK sequence.<br>Number of QUERYREPs is determined by the global Q value.<br>No SELECT is sent, so no masking occurs, even with masks active. | No SELECT<br>No XDATA |
| **T2** | T2[<LOOPCOUNT>]<br>Same as T1, but each tag reports the RF frequency it was acquired on.<br>No SELECT is sent, so no masking occurs, even with masks active.<br>No XDATA processing occurs, even with XDATA DESCRIPTORs active. | No SELECT<br>No XDATA |
| **T3** | T3[<LOOPCOUNT>]<br>Sends a SELECT/QUERY/QUERYREP/ACK sequence.<br>Number of QUERYREPs is determined by the global Q value.<br>No XDATA processing occurs, even with XDATA DESCRIPTORs active. | No XDATA |
| **T4** | T4[<LOOPCOUNT>]<br>Same as T3, but each tag reports the RF frequency it was acquired on.<br>No XDATA processing occurs, even with XDATA DESCRIPTORs active. | No XDATA |
| **T5** | T5[<LOOPCOUNT>]<br>Same as T3, but XDATA processing occurs for each tag found.<br>This adds REQRN/READ and/or WRITE commands. | - |
| **T6** | T6[<LOOPCOUNT>]<br>Same as T5, but each tag reports the RF frequency it was acquired on. | - |

## "T" and "T<n>" Command Examples

### Basic "Get Tags"

```
READY>t5
STARTINVENTORY
TAG=3000E2003411B801010861355058
TAG=3000BAD1000000000000000000000
TAG=3000E2003412DC03011827047484
STOPINVENTORY 0x0221 EPCCOUNT=3
```

### Get Tags, Including EPC DATA

```
READY>t6
STARTINVENTORY
TAG=3000E2003411B801010861355058 923250 00 02 01 06 3 FB66
TAG=3000BAD1000000000000000000000 923250 00 02 02 02 2 FB82
TAG=3000BEEF000000000000000000006 923250 00 02 09 00 1 FBAE
STOPINVENTORY 0x00FA EPCCOUNT=3
```

### Perform a Continuous T1

```
READY>t1
STARTINVENTORY
TAG=3000BAD1000000000000000000000
TAG=3000E2003411B801010861355058
TAG=3000BAD1000000000000000000000
TAG=3000E2003411B801010861355058
TAG=3000E2003411B801010861355058
TAG=3000E2003411B801010861355058
TAG=3000BAD1000000000000000000000
TAG=3000E2003411B801010861355058
TAG=3000E2003411B801010861355058
TAG=3000E2003411B801010861355058
TAG=3000E2003411B801010861355058
TAG=3000E2003411B801010861355058
TAG=3000E2003411B801010861355058
<key pressed>
STOPINVENTORY 0x0125 EPCCOUNT=14
```

### Perform a Single T2

```
READY>t21
STARTINVENTORY
TAG=3000E2003411B801010861355058 906750 07 9 6 I D3C8
TAG=3000BAD1000000000000000000000 906750 01 5 0 I D3D9
STOPINVENTORY 0x001F EPCCOUNT=2
```

### Perform Four T6s

```
READY>t64
STARTINVENTORY
TAG=3000E2003411B801010861355058 908250 05 A 6 I D3B8
TAG=3000BAD1000000000000000000000 908250 02 5 0 I D3C5
TAG=3000E2003411B801010861355058 908250 01 A 6 I D3DF
```

```
TAG=3000BAD10000000000000000000 908250 00 5 0 I D3EB
TAG=3000BAD10000000000000000000 908250 03 5 0 I D3FC
TAG=3000E2003411B801010861355058 908250 02 A 7 Q D409
STOPINVENTORY 0x0062 EPCCOUNT=6
```

## ***Calculating Signal Strength (RSSI) from the I/Q Magnitude Fields***

Tag data returned from a Tn inventory (where n= 2,4,6) include fields for I and Q signal magnitude. You can use these fields to calculate an overall signal strength for the read that can give you some indication of the range of the tag to the antenna.

In desktop applications like programming, this is especially useful to discriminate between a tag that is right next to the antenna vs. one some distance away. You may choose to filter the data reported to an end user of your application by signal strength to only show nearby tags. – One of the example programs provided by Thinkify in the TR200 developer's kit does just this.

Recall the magnitudes are delivered as part of a tag read message:

```
TAG=3000E2003411B801010861355058 908250 02 A 7 Q D409
```

(Here the I channel magnitude is A (decimal 10) and the Q channel magnitude is 7.)

To calculate the signal strength, use the following relationship:

```
rssi = 2 * high_rssi + 10 * Log(1 + 10 ^ (-delta_rssi / 10))
```

Where:

```
high_rssi = 10
```

(the larger RSSI is the I channel with a value of A.(10 decimal)) and

```
delta_rssi = Abs(imag - qmag)
delta_rssi = 3
```

## "X" – eXtra Data Read and Write Descriptor Control

```
X[<R|W>[<PARAMS>]]
```

Anytime an EPC code is acquired from a tag, an opportunity exists to either read additional data from the tag, or write data to it. These options are controlled by XDATA descriptors managed by the X commands.

The TR265 reader maintains four (4) XDATA read descriptors and four (4) XDATA write descriptors that may be individually configured to perform read/write operations.

By default all XDATA descriptors are disabled. When a tag's EPC is decoded, each of the XDATA descriptors are checked for an ACTIVE status, which causes a read/write at the specified location to be performed. Inside an inventory mode which supports XDATA (currently T, T5, and T6) the operations will be performed right after the read of the EPC data, and the data appears on the line of output immediately following the EPC data in the tag stream.

### Flags

`<PARAMS>` may contain some or all of the following:

| | | |
|---|---|---|
| `<#>` | – | Descriptor number |
| `<ACTIVE>` | – | Descriptor enabled |
| `<MEMBANK>` | – | Tag memory bank for the operation |
| `<LEN>` | – | Length (in words) of data to be read/written |
| `<EBV>` | – | EBV pointer into memory for the start of the operation |
| `<DATA>` | – | The bytes of data to be written. |

### Sub-Commands

| Sub Command | Description | Legal Values for SET |
|---|---|---|
| **XR** | Report all XDATA read descriptors. | - |
| **XRR** | Reset all XDATA read descriptors. | - |
| **XR<#>** | Report a given XDATA read descriptor. | 0..3 |
| **XR<#><ACTIVE>** | Set the <ACTIVE> flag for XDATA read descriptor <#>. | 0..1 |
| **XR<#><ACTIVE><...>** | Configures XDATA read descriptor <#> to perform a read at the specified location and length.<br>`XR<#><ACTIVE><MEMBANK><LEN><EBV>`<br>`<ACTIVE>  0=inactive, 1=active`<br>`<MEMBANK> 0..3` | See Description |

| Sub Command | Description | Legal Values for SET |
|---|---|---|
| | `<LEN>     1..8 (# of words to read)`<br>`<EBV>     Word pointer into memory`<br>`          (1-4 bytes)` | |
| **XW** | Report all XDATA write descriptors. | - |
| **XWR** | Reset all XDATA write descriptors. | - |
| **XW<#>** | Report a given XDATA read descriptor. | 0..3 |
| **XW<#><ACTIVE>** | Set the <ACTIVE> flag for XDATA write descriptor <#>. | 0..1 |
| **XW<#><ACTIVE><...>** | Configures XDATA write descriptor <#> to perform a write at the specified location, length, and with data provided.<br>`XW<#><ACTIVE><MEMBANK><LEN><EBV><DATA>`<br>`<ACTIVE>`<br>`BIT0 - USE DESCRIPTOR`<br>`BIT1 - Change PC if length different from tags current length`<br>`BIT2 - USE BLOCKWRITE`<br>`BIT3 - INCREMENT DESCRIPTOR DATA after successful write.`<br>`<MEMBANK> 0..3`<br>`<LEN>     1..8 (# of words to write)`<br>`<EBV>     Word pointer into memory (1-4 bytes)`<br>`<DATA>    Data to write to location` | See Description |

## *"X" Command Examples*

### *Read Extra Data in T Command*

```
// Do inventory with default parameters.
READY>t51
STARTINVENTORY
TAG=3000E2003411B802011029356733
STOPINVENTORY 0x0001 0x004A

// Set read descriptor #0 to read
// Bank:1, Len:4, WordPntr:2
READY>xr011402
RDDESCRIPTOR=0
ACTIVE=1
BANK=1
LEN=4
PNTR=02

// Look for the extra data
```

```
READY>t51
STARTINVENTORY
TAG=3000E2003411B802011029356733
XRD0=E2003411B8020110
STOPINVENTORY 0x0039 EPCCOUNT=1
```

### *Read Extra Data in T<n> Command*

```
// Do 10 (0xA) iterations of T6
READY>t6A
STARTINVENTORY
TAG=3000E2003411B802011029356733 924250 05 E B I 1FBF
XRD0=E2003411B8020110
TAG=3000E2003411B802011029356733 926750 00 E C Q 1FF0
XRD0=E2003411B8020110
TAG=3000E2003411B802011029356733 926750 02 E C Q 2007
XRD0=E2003411B8020110
TAG=3000E2003411B802011029356733 926750 06 E C I 201E
XRD0=E2003411B8020110
TAG=3000E2003411B802011029356733 926750 02 E C I 2038
XRD0=E2003411B8020110
TAG=3000E2003411B802011029356733 926750 06 E C Q 204F
XRD0=E2003411B8020110
TAG=3000E2003411B802011029356733 926750 05 E C Q 2068
XRD0=E2003411B8020110
TAG=3000E2003411B802011029356733 926750 03 E C I 2081
XRD0=E2003411B8020110
STOPINVENTORY 0x00DB EPCCOUNT=8
```

### *Set a Write Descriptor, Then Get It*

```
READY>xw011402111122223333444
WRDESCRIPTOR=0
ACTIVE=1
BANK=1
LEN=4
PNTR=02
WRITEDATA=1111222233334444

READY>xw0
WRDESCRIPTOR=0
ACTIVE=1
BANK=1
LEN=4
PNTR=02
WRITEDATA=1111222233334444
```

### *Set and Use a Write Descriptor*

```
// 1st read a tag
READY>t5
STARTINVENTORY
TAG=3000E2003411B802011029356733
STOPINVENTORY 0x0034 EPCCOUNT=1

// Set up to rewrite part of the EPC
// Bank:1, WordPntr:02, Len:03,
// Data:AABBCCDDEEFF
READY>xw011302AABBCCDDEEFF
WRDESCRIPTOR=0
ACTIVE=1
BANK=1
LEN=3
PNTR=02
WRITEDATA=AABBCCDDEEFF

// Read the tag again
// (automatically performs the write).
READY>t51
STARTINVENTORY
TAG=3000E2003411B802011029356733
XRD0=SUCCESS
STOPINVENTORY 0x005E EPCCOUNT=1

// Read again and we see the new EPC
READY>t51
STARTINVENTORY
TAG=3000AABBCCDDEEFF011029356733
XRD0=SUCCESS
STOPINVENTORY 0x003C EPCCOUNT=1
```

### ***Using LOOPCOUNT to Retry Writes***

You can use a T6 inventory command with LOOPCOUNT of 0xA (10 loops) to perform a write. The WRITE success operation is given when all data matches the requested write field. Once the data matches all XWR messages will indicate success with no further actual write attempts.

Any XREAD or XWRITE that does not complete successfully returns an error code. Some portion of a WRITE operation may complete and still return an error code, if multiple word writes are requested. Also note that in the case of a WRITE, an error code is generated if the ASYNC response from the tag is improperly decoded, although the WRITE may have actually worked.

```
READY>xw0114021111222233334444
WRDESCRIPTOR=0
ACTIVE=1
BANK=1
LEN=4
PNTR=02
WRITEDATA=1111222233334444

READY>t610
STARTINVENTORY
// First inventory loop
TAG=3000AAAABBBBCCCC011029356742 919750 07 C E Q CB2D
XRD0=SUCCESS
// Next loop shows new id.
TAG=30001111222233334444429356742 919750 05 C E I CB83
XRD0=SUCCESS
TAG=30001111222233334444429356742 919750 00 C E I CBC5
XRD0=SUCCESS
TAG=30001111222233334444429356742 919750 07 C E I CBE4
XRD0=SUCCESS
```

```
TAG=3000111122223333444429356742 919750 03 C E Q CC07
XRD0=SUCCESS
TAG=3000111122223333444429356742 919750 01 C E I CC29
XRD0=SUCCESS
TAG=3000111122223333444429356742 919750 00 C E I CC4E
XRD0=SUCCESS
STOPINVENTORY 0x014F EPCCOUNT=7
```