RFID 7204

PROTOCOL REFERENCE GUIDE

v1.0

neology
mobility [re]imagined.

avahi-daemon-0.7-r0
avahi-locale-en-gb-0.7-r0
base-files-3.0.14-r89
base-passwd-3.5.29-r0
bash-4.4.18-r0
bridge-utils-1.6-r0
busybox-1.29.3-r0
bzip2-1.0.6-r5
connman-1.35-r0
crda-3.18-r0
cronie-1.5.2-r0
daemontools-0.76-r0
db-5.3.28-r1
dbus-1-1.12.10-r0
dropbear-2018.76-r0
eudev-3.2.7-r0
eudev-hwdb-3.2.7-r0
firmware-imx-epdc-7.8-r0
firmware-imx-vpu-imx6d-7.8-r0
firmware-imx-vpu-imx6q-7.8-r0
fsl-rc-local-1.0-r0
glibc-locale-en-gb-2.28-r0
gpgme-1.11.1-r0
gptfdisk-1.0.4-r0
grep-3.1-r0
hdparm-9.56-r0
imx-kobs-5.5+git0+a0e9adce2f-r0
inetutils-1.9.4-r0
init-ifupdown-1.0-r7
initscripts-functions-1.0-r155
libncursesw5-6.1+20180630-r0
libnetsnmp35-5.8-r0
libnetsnmpagent35-5.8-r0
libnetsnmphelpers35-5.8-r0
libnetsnmpmibs35-5.8-r0
libnetsnmptrapd35-5.8-r0
libnl-3-200-3.4.0-r0
libnl-genl-3-200-3.4.0-r0
libnss-mdns-0.10-r7
libpanelw5-6.1+20180630-r0
libpci3-3.6.2-r0
libpcre1-8.42-r0
libperl5-5.24.4-r0
libpopt0-1.16-r3
libpython2.7-1.0-2.7.15-r1
libreadline7-7.0-r0
libsqlite3-0-3.23.1-r0
libssl1.1-1.1.1a-r0
libstdc++6-8.2.0-r0
libtinfo5-6.1+20180630-r0
libtirpc3-1.0.3-r0
libts0-1.16-r0
libudev1-3.2.7-r0
libunistring2-0.9.10-r0
libusb-1.0-0-1.0.22-r0
libuuid1-2.32.1-r0
libwrap0-7.6-r10
libxml2-2.9.8-r0
libxslt-1.1.32-r0
libz1-1.2.11-r0
linux-firmware-imx-sdma-imx6q-0.0+git0+linux+firmware-r0
linux-firmware-imx-sdma-license-0.0+git0+linux+firmware-r0
locale-base-en-gb-2.28-r0
locale-base-en-us-2.28-r0
mdns-765.50.9-r0
modutils-initscripts-1.0-r7
ncurses-terminfo-base-6.1+20180630-r0
netbase-5.4-r0
net-snmp-5.8-r0
nettle-3.4-r0
nginx-1.15.2-r0
ntp-4.2.8p12-r0
ntp-tickadj-4.2.8p12-r0
openssl-1.1.1a-r0
packagegroup-base-extended-1.0-r83
packagegroup-base-ipv6-1.0-r83
packagegroup-base-nfs-1.0-r83
packagegroup-base-1.0-r83
packagegroup-base-usbgadget-1.0-r83
packagegroup-base-usbhost-1.0-r83
packagegroup-core-boot-1.0-r17
packagegroup-core-ssh-dropbear-1.0-r1
packagegroup-distro-base-1.0-r83
packagegroup-machine-base-1.0-r83
perl-5.24.4-r0
pm-utils-1.4.1-r1
pointercal-0.0-r11
psplash-default-0.1+git0+2015f7073e-r15
psplash-0.1+git0+2015f7073e-r15
python-3.8.3
rng-tools-5-r0
rpcbind-0.2.4-r0
run-postinsts-1.0-r10
shadow-base-4.6-r0
shadow-securetty-4.6-r3
shadow-4.6-r0
shared-mime-info-1.10-r0
sysvinit-inittab-2.88dsf-r10
sysvinit-pidof-2.88dsf-r14
sysvinit-2.88dsf-r14
tslib-calibrate-1.16-r0
tslib-conf-1.16-r0
tslib-tests-1.16-r0
tzdata-2018i-r0
u-boot-variscite-1.0-r0
udev-rules-imx-1.0-r0
update-alternatives-opkg-0.3.6-r0
update-rc.d-0.8-r0
usbutils-010-r0
util-linux-sulogin-2.32.1-r0
uwsgi-2.0.19.1
vsftpd-3.0.3-r0
xinetd-2.3.15-r2
zip-3.0-r2

12760 Danielson Court, Suite A

Poway, CA 92064

PREFACE

Intended audience

The *Neology Toll RFID Reader* 7204 Protocol Reference Guide is designed for users and developers who need to configure applications to communicate with Neology Toll RFID Reader 7204s.

What's in this guide

The information in this guide is presented as follows:

Introduction

This chapter describes the various ways you can connect to the reader. The human and machine interfaces are described along with methods for connecting to the communication ports.

Overview of the Neology Toll RFID Reader 7204 User Interface

Describes how to establish a connection with the readers Command Line Interface (CLI). Various methods are described, such as the Secure Shell (SSH), terminal emulation, and machine connections.

Overview of the Neology Toll RFID Reader 7204 Commands

This chapter provides a brief overview of the namespace commands used by the 7204. In addition, the data types and types of logins are described.

Reader Behavior

This chapter describes reader profiles, reader setup, antenna configuration, and digital I/O configuration.

Embedded Reader Applications

This chapter describes how to load embedded applications such as Python scripts.*

Tag Protocol Support

This chapter describes protocol specific conventions and reader setup requirements.

Namespace Chapters

These chapters list all the functions, variables, events, and errors for the 7204.

Interpreting Reader Logs

This appendix provides information for interpreting reader error logs.

Conventions used in this manual

The following conventions are used in this manual:

Bold courier	font indicates code entered by the user
(values)	within parentheses indicate parameters
(*values*)	in italics indicate user defined variables.
<n>	indicates a variable number used in a function that can apply to several different devices such as antennas or I/O ports.
WARNING	Indicates a hazardous situation which, if not avoided, could result in death or serious injury.
CAUTION	Indicates a hazardous situation which, if not avoided, could result in minor or moderate injury or property damage.
IMPORTANT	This yellow symbol indicates that the device is susceptible to Electro Static Discharge and appropriate precautions must be taken to avoid equipment damage.
NOTE	advises the reader that a condition can be created by a particular action that can cause equipment damage or result in equipment operation that violates regulatory requirements.

CONTENTS

CONTENTS

1 INTRODUCTION

1 INTRODUCTION

1.1 CONTROLLING THE NEOLOGY TOLL RFID READER 7204

The Neology Toll RFID Reader 7204 is a highly versatile RFID system that can be configured to operate in most Automatic Vehicle Identification (AVI) RFID applications. As a result, all reader management functions and parameters are easily configured by sending specific Command Line Interface (CLI) commands to the reader's command channel. These commands are organized using **namespaces**.

A namespace is a context for identifiers and is usually written in human readable form. A configuration namespace provides organization and hierarchy to a set of configuration variables. Namespaces can be nested, with multiple namespaces existing under a higher namespace.

As shown by this example, any identifier defined within a namespace is associated with the namespace and the same identifier can be used within different namespaces. This prevents conflicts within the namespaces themselves and allows the namespaces to nest forming a namespace tree. The root of the tree is always the unnamed global namespace.

Namespaces

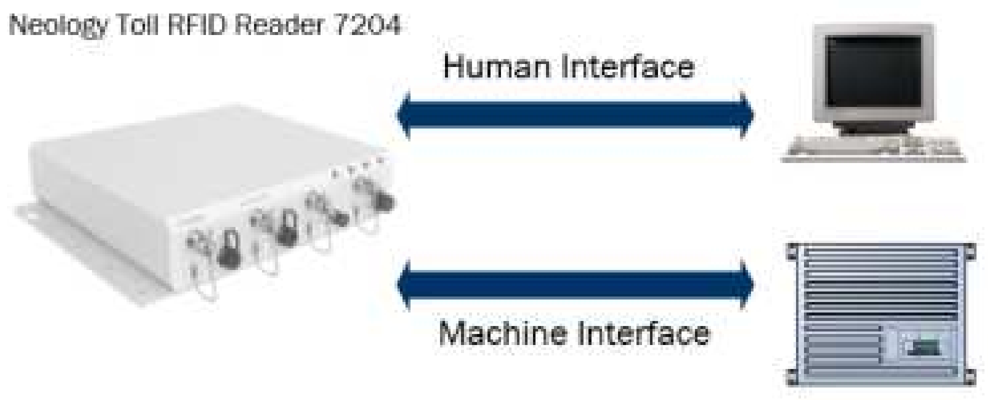
A namespace is an abstract container which is filled by names that represent or stand for configuration items or actions. As a rule, names in a namespace cannot have more than one meaning, that is, two or more things cannot share the same name. A namespace is also called a context, as the valid meaning of a name can change depending on what namespace applies

Top-level namespaces are provided for the following:

- **Reader** – Generic reader level functions
- **Setup** – Reader setup
- **Info** – Reader information
- **Version** – Hardware and software versions
- **Com** – Communication level features
- **Tag** – Tag control features
- **DIO** – Digital input and output features
- **Antennas** – Antenna configuration
- **Modem** – Low-level modem control
- **User** – User defined variables

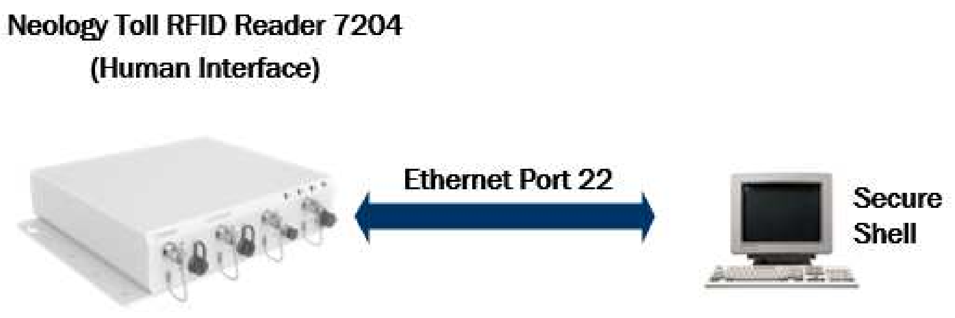
1.2 TYPES OF INTERFACES

The Neology Toll RFID Reader 7204 supports human and machine communication across either the serial or Ethernet ports. For the purposes of the human interface discussion, this manual assumes you are communicating with the reader using a standard Windows based PC.



1.2.1 Human Interface

The human interface allows you to manually submit commands to the reader and observe any responses. There are several methods you can use to establish this type of interface: Secure Shell (SSH) or Terminal Emulation.



To communicate across the Ethernet port, open a SSH session using an application such as PuTTY. This is a free application that is available from

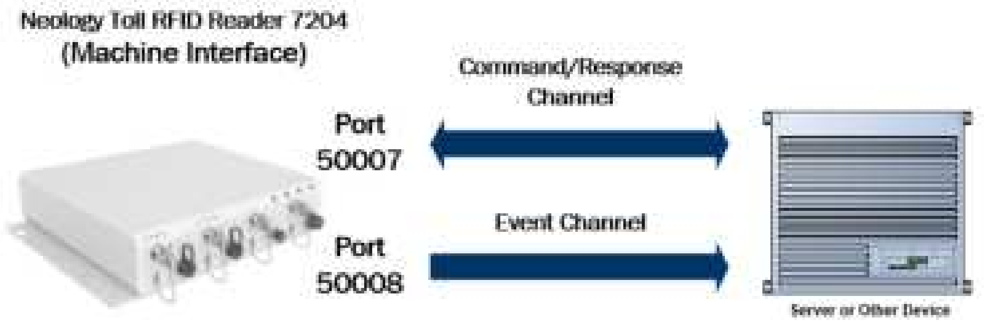
PuTTY SSH Client

You can use any SSH client application to establish an SSH reader connection. Examples in this guide use a free application called PuTTY. This application is distributed under the MIT license and is provided on the product CD.

1.2.2 Machine Interface

Command/Response and Event Channels

The machine interface is a direct connection to the reader's command/response and event channels. The command/response channel is a bidirectional interface and connects across port `50007`. Commands and the corresponding responses are sent across this channel. The event channel is unidirectional and connects across port `50008`. The event channel reports asynchronous events such as errors, tag arrivals, and digital output triggers.



A sample of software code required to open a communication session with a reader is provided in the *Establishing a Machine Interface* section in Chapter 2.

Interface Protocol

The Neology Toll RFID Reader 7204's native protocol is ASCII text based and is used to communicate over the Command/Response and Event Channels.

Every command is sent to the reader over port `50007` and should end with `\r` and `\n`.

`\r` and `\n` are non-printable carriage return and line feed escape sequences.

The corresponding reader response will end with `\r\n\r\n`.

An example of a command and response is shown in the following:

```
-setup.region\r\n
->fcc\r\n\r\n
```

Events are returned from reader over port 50008. Every event is completed with `\r\n\r\n`. An example event is shown in the following:

```
event.tag.report
tag_id=0xAABBCCDDEEFFGG\r\n\r\n
```

Secure Channels

The Neology Toll RFID Reader 7204 provides encrypted request and response channels for secure communication.

Secure access to the ASCII command line interface is provided via TLS/SSL encryption. There are no usage differences in the ASCII command line interface when using encrypted or non-encrypted channels. For TLS/SSL, the following ports are utilized:

- Port `50002` (command and response channel)
- Port `50003` (event channel)

Secure access to the reader discovery service is provided via PGP encryption. The reader discovery responses will be PGP encrypted if the incoming message was PGP encrypted and successfully decoded.

For PGP, the following ports are utilized:

- Port `50000` (discovery request channel)
- Port `50001` (discovery response channel)

2 USER INTERFACE

2 USER INTERFACE

2.1 ESTABLISHING A HUMAN INTERFACE

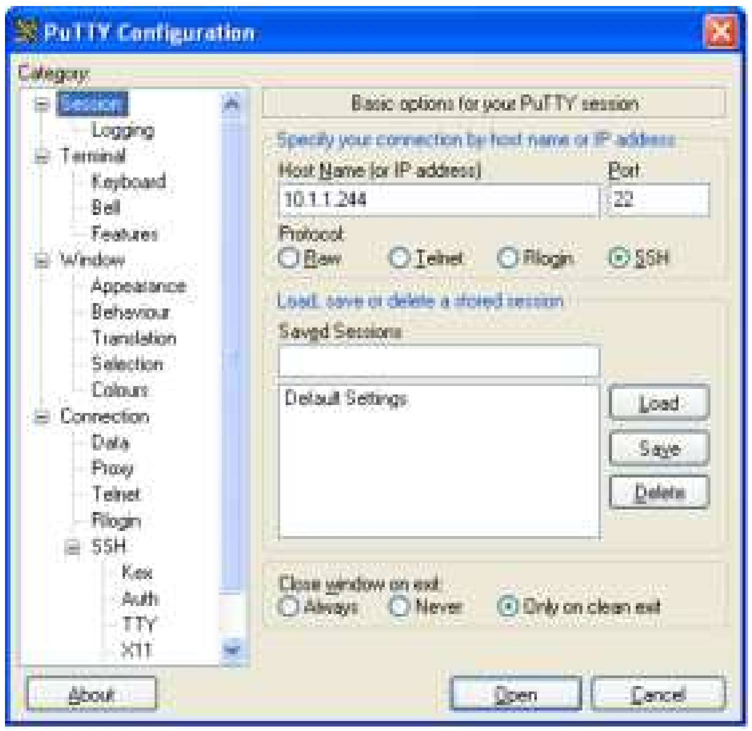
As described in Chapter 1, establishing a Human Interface with the Neology Toll RFID Reader 7204 uses a Secure Shell (SSH) session. This section chapter describes how to open this connection.

2.1.1 Connect by SSH

Before establishing an SSH connection with the reader, you must first obtain the IP address of the reader. To obtain the IP address,use the Reader Startup Tool (RST). Refer to the Neology Toll RFID Reader 7204 User’s Guide for information on using RST.

Establish SSH Client

1 Establish an SSH client with PuTTY or similar application.



Host Name (or IP address): 10.1.1.244

Port: 22

Protocol: Raw Telnet Rlogin [✓ SSH]

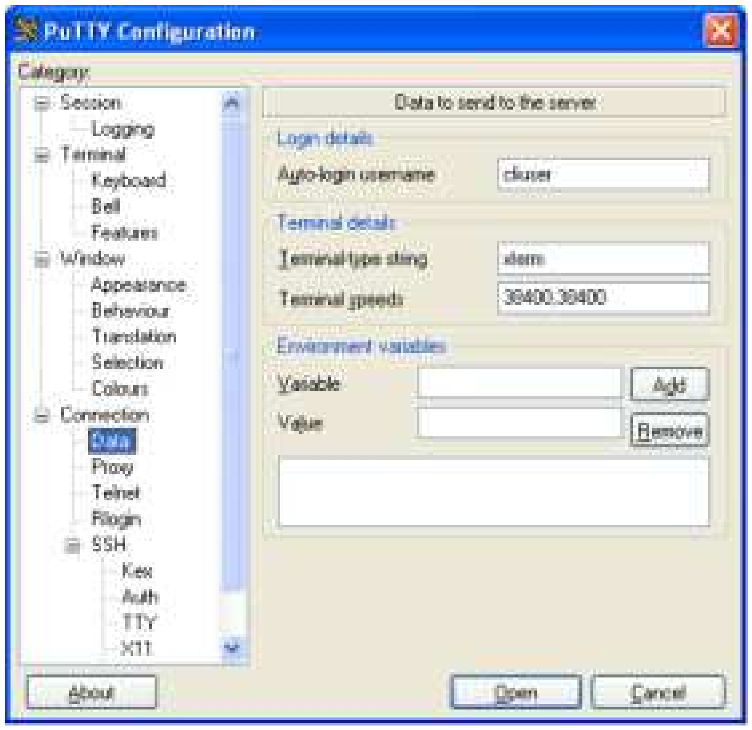
Saved Sessions:

Default Settings

Close window on exit: Always Never [✓ Only on clean exit]

2 Enter the IP address of the reader.

3 Press **Open**.



Data to send to the server

Login details

Auto-login username: cliuser

Terminal details

Terminal-type string: xterm

Terminal speeds: 38400,38400

Environment variables

Variable:

Value:

4 Select **Data** in the tree on the left and enter: **cliuser** in the **username** field.

5 Do not enter a password.

6 Press **Open** and the Command Line Interface (CLI) becomes available.

7 Enter commands at the prompt.

2.2 ESTABLISHING A MACHINE INTERFACE

To establish a machine interface with the Neology Toll RFID Reader 7204, open a Socket Connection using the IP address of the reader. Use the Discovery Tool provided on your product CD to obtain the IP address. The Discovery procedure is described in the Neology Toll RFID Reader 7204 User's Guide. The following `C# .net` code opens a socket and starts reading and writing to the reader.

```
{
  IPAddress
f_IPAddress=IPAddress.Parse(m_IPAddress);


  // Command port runs on 50007
  IPEndPoint f_IPEndPoint = new
IPEndPoint(f_IPAddress,
  Int32.Parse(“50007”));
  Socket m_Clientsocket = new
Socket(f_IPEndPoint.AddressFamily,
  SocketType.Stream, ProtocolType.Tcp);
  m_Clientsocket.Connect(f_IPEndPoint);

  // Check if socket is connected
  if (m_Clientsocket.Connected==true)  {

    // Create Reader and Writer objects to
perform higher level calls.
    socketStream = new
NetworkStream(m_Clientsocket);
    StreamReader streamReader = new
StreamReader(socketStream);
    StreamWriter streamWriter = new
StreamWriter(socketStream);
    streamWriter.AutoFlush = true; // enable
automatic flushing

    // Send command and receive response
    streamWriter.WriteLine(f_Command);
    string responseValue =
streamReader.ReadLine();
    string readdata =
streamReader.ReadLine();
    string responseValue=readdata;
    while(!readdata.Trim().Equals("")) {
      readdata = streamReader.ReadLine();
      responseValue += readdata;
    }
```

```
        // Perform action on response.
        // Customer puts action code here.
        // Close Connection
        socketStream.Close();
    } else {
        throw new ConnectionException("Cannot
connect to Reader.");
    }

    //Close Connection
    m_Clientsocket.Close();
}
    catch(Exception exc) {
        throw new
ConnectionException(exc.Message);
    }
```

3 COMMANDS

3 COMMANDS

3.1 COMMAND CATEGORIES

Neology Toll RFID Reader 7204 commands are categorized based on the following actions:

- **Get** - Retrieves the value of a specified reader configuration variable.
- **Set** - Sets the value of the specified reader configuration variable.
- **Exec** - Executes a function in the reader.

3.1.1 Get

A **Get** action retrieves the value of a specified reader configuration variable. The response is either **OK** or an error message.

The syntax of a Get command is as follows:

```
variable_name
```

In the following example, the `com.network.1.ip_address` variable is retrieved:

```
>>> com.network.1.ip_address
ok 192.168.68.34
```

3.1.2 Set

A **Set** action sets the value of the specified reader configuration variable. The syntax of a Set command is as follows:

```
variable_name = value
```

In the following example, the `setup.protocols` variable is set to `isoc`:

```
>>> setup.protocols=isoc
ok
```

3.1.3 Exec

An **Exec** action executes a reader function. The syntax of a Exec command is as follows:

```
function_name(name1=value1, name2=value2)
```

In the following example, the `tag.write` command writes a value of `0x12345678` to user data in a tag with an EPC of `0x30112233445566778899aabb`:

```
>>> tag.write_user_data(user_data=0x12345678,
tag_id=0x30112233445566778899aabb)
ok
```

3.2 COMMAND RESPONSES

As shown in the previous section, the reader supports three types of commands: **Get**, **Set** and **Exec**. The response to all commands is either **OK** or an error message.

3.2.1 Get Command Response

For a Get command the successful response is **ok** followed by the value of the variable. For example, if the `com.network.hostname` variable has the value `HostName`, the command to get the value of this variable and its response would be:

```
>>> com.network.hostname
ok HostName
```

3.2.2 Set Command Response

For a set command the successful response is simply **ok**. For example, to set the value of the `info.location` variable to `Toll Zone 3`, the command and its response would be:

```
>>> info.location = Toll Zone 3
ok
```

3.2.3 Exec Command Response

An Exec action executes a reader function. For an Exec command, the successful response is **ok** followed by the data returned by the function. The data returned is different for each function. For example, to exec (execute) the function `tag.read_id()` to obtain the tag id of a tag in the field, the command and its response would be:

```
>>> tag.read_id()
ok tag_id=0x300102030405060708090a0b
```

3.2.4 Error Responses

In the event a command fails, an error is returned. For example, setting the value of the `tag.reporting.depart_time` to `1` will produce the `error.parser.value_out_of_range` response because the range of acceptable values for `tag.reporting.depart_time` is from `100` to `25000` milli-seconds.

```
>>> tag.reporting.depart_time = 1
error.parser.value_out_of_range
```

The errors returned by the reader are contained in the error namespace. All possible errors in the error name space are shown in the Errors Namespace chapter.

3.3 DATA TYPES

7204 variable and function parameters use the following data types:

Data Type	Description	Description/Example
`Bool`	True/false	True or False
`String`	any string	abcdefghijklmnopqrstuvwxyz1234567890
`Int`	Numbers	1234567890
`Enum`	Any one enum value	If Enum options are FCC or ETSI, user can pass either one.
`Enumlist`	Multiple values from enum list	"ISOC ISOB"
`Array`	Series of hex values	"aabbccddeeff112233"
`List`	Array of integers	"900 901 902 903"
`Compound`	Structure	tagid, userdata, lock_code, kill_code
`Compoundlist`	List of structures	Taglist will be multiple entries for tag structure.

3.4 PERMISSIONS AND READER PASSWORDS

The 7204 supports an authentication scheme in which users must supply a valid login level and password to access the reader. Each login level provides different permissions to access the reader's features. The reader also supports a feature to set a default level that can be assigned to all clients that connect to the reader.

3.4.1 User Level

The reader's login levels are:

- guest
- admin

The guest login level provides read-only access to the reader. Clients that login at the guest level can read the settings of the reader and can access the tags that the reader has inventoried. Clients at this level cannot change the configuration of the reader.

The admin login level provides read-write access to the reader. Clients that login at the admin level can read and write the settings of the reader and can access the tags that the reader has inventoried.

To assign the login levels, each namespace command has one or more of the following permissions:

- `r` = read
- `w` = write
- `x` = execute
- `-` = not allowed

These permissions are provided for each command in the following format:

guest=`-`, admin=`x`

3.4.2 Initial Passwords

The reader is delivered with the following passwords for the login levels:

Login Level	Initial Password
guest	readerguest
admin	readeradmin

3.4.3 Changing Passwords

The `reader.set_pwd()` command is used to change the passwords. This command requires the following three parameters:

- `login` (level for password to be changed)
- `pwd` (existing password for the login level)
- `new_pwd` (new password for the login level)

The login and existing password are required to change the password. For example, to change the initial password for the guest login level to **19qht34** use the following command:

```
>>>
reader.set_pwd(login=guest,new_pwd=19qht34)
ok
```

3.4.4 Setting the Default Login Level

The reader supports a default login level to accommodate multiple access levels. The default login level is the level assigned to each client as it connects to the reader. The default login level can be set to:

`unauthenticated`	When the default login level is set to unauthenticated, all clients that connect to the reader must login (with the `reader.login()` command) before they can access any features of the reader. Only the `reader.login()` command is accepted when the default login level is set to unauthenticated. This allows the reader to be locked down and forces all connecting clients to login before accessing the reader.
`guest`	When the default login level is set to guest, all clients that connect to the reader are automatically logged in at the guest level at connection time. This means they can perform all guest permitted actions without issuing the reader.login() command.
`admin`	When the default login level is set to admin, all clients that connect to the reader are automatically logged in at the admin level at connection time. This means they can perform all admin permitted actions without issuing the reader.login() command. The factory default login level is admin. This allows for a quick turn-up and initial testing of the reader. When the reader is deployed, set the default login level to the level that best suits your security/accessibility needs.

To set the default login level, use the `setup.default_login_level` configuration variable. For example to set the default login level to guest use the following command:

```
>>> setup.default_login_level = guest
ok
```

3.4.5 Embedded Application/Script Login

You can customize your reader by writing applications and scripts that execute on the reader. When an application or script executes, it connects as a client. When an application or script running on the reader connects, it is automatically logged in at the admin level. This allows applications and scripts to run on any reader without knowing the password.

4 READER BEHAVIOR

4 READER BEHAVIOR

4.1 READER PROFILES

The reader's configuration is stored in a profile. A profile contains the setting of all the configuration variables in the reader. You can perform the following profile functions:

- Get the name of the active profile
- Create new profiles
- List all profiles
- Load a profile
- Delete a profile

The reader can store up to 8 different profiles.

4.1.1 Profile Name Conventions

Profile names must conform to the following rules:

- Consist of characters A – Z, a – z, 0 – 9, '-' or '_'
- Must be between 1 and 32 characters in length

4.1.2 Factory Default Profile

The reader comes with a read-only profile called **factory**. Initially, readers are configured to load the factory profile at startup. This profile can always be loaded with the `reader.profile.reset_factory_default()` command followed by a `reader.reboot()`.

4.1.3 Active Profile

The active profile is the profile that is currently used to configure the reader. The name of the active profile is held by the `reader.profile.active` read-only configuration variable. To get the name of the active profile, simply get the value of the reader.profile.active variable. The following example shows the active profile is named `isoc_vehicle`.

```
>>> reader.profile.active
ok isoc_vehicle
```

4.1.4 Creating a New Profile

New profiles are created by saving the current configuration of the reader after any desired configuration changes are made. The current configuration can be saved as a new profile by executing the `reader.profile.save()` command.

To illustrate this concept, assume the reader's active configuration is the factory default. To create a profile that is setup to read ISOC tags from a vehicle, change the `setup.protocols` configuration variable to indicate that the reader should inventory ISOC tags. This command is:

```
>>> setup.protocols = ISOC
ok
```

The reader is now configured to inventory ISOC tags. A new profile can now be saved with the `reader.profile.save()` command. To save the current configuration use the following:

```
>>> reader.profile.save(isoc_vehicle)
ok
```

Once a profile is saved, it becomes the active profile. To get the active profile after the save, use the following:

```
>>> reader.profile.active
ok isoc_vehicle
```

This response shows the active profile is `isoc_vehicle`. When a profile is saved it becomes the startup profile, the profile the reader loads the next time it is started. As stated previously, the reader can store up to 8 different profiles.

4.1.5 Listing all Profiles

The list of profiles on the reader can be obtained using the `reader.profile.list()` command. This command lists all profiles saved on the reader. For example:

```
>>> reader.profile.list()
ok profiles isoc_vehicle isoc_parking
```

4.1.6 Loading a Profile

Once profiles are created with the `reader.profile.save()` command, they can be loaded with the `reader.profile.load()` command. To switch the reader profile, use the following command:

```
>>> reader.profile.load(isoc_vehicle)
ok
```

When a profile is loaded, the configuration settings in that profile are immediately applied to the reader and that profile becomes the startup profile. The startup profile is the profile the reader loads the next time it is started.

4.1.7 Deleting a Profile

A profile can be removed from the reader with the `reader.profile.delete()` command. This command removes the specified profile from the reader. Care must be taken with this command, once a profile is deleted, it cannot be recovered. To delete a profile, first list the profiles:

```
>>> reader.profile.list()
ok profiles isoc_vehicle isoc_parking
```

To delete the profile `isoc_parking`, use the following command:

```
>>> reader.profile.delete(isoc_parking)
ok
```

If the deleted profile was the startup profile, the **factory** profile becomes the startup profile.

4.2 READER SETUP

The reader provides a small set of high level configuration variables that are used to ease the initial configuration process. These configuration variables reside in the **setup** namespace and control basic properties of the reader. These basic properties are used to derive the entire detailed configuration set for the reader.

When you first receive your reader, use the **setup** variables to initially configure the reader for a general application and then customize a few of the specific settings. In most cases, the detailed configuration automatically applied by the reader will be sufficient for effective use of the reader.

The following user settable variables exist in the **setup** namespace and determine the basic properties of the reader:

1 Region
2 Sub-region
3 Protocols
4 Install Type
5 Tag Volume
6 Operating Mode

Always set these variables in the order shown in the previous list.

4.2.1 Region

The `setup.region` variable sets the regulatory mode for the geographic region where the reader is deployed. The options are factory locked, such as those following the FCC regulatory standards, and have a fixed value for this setting that cannot be modified.

4.2.2 Sub-Region

The `setup.sub_region` variable sets the secondary regulatory mode for geographic region where the reader is deployed. You can modify this setting, if there is more than one option available under the current region setting. Possible values are in the `setup.valid_sub_regions` variable.

In the **FCC** region, there are two different sub-region settings for Part 90 operation. The first sub-region (FCC_PART90_DENSE) utilizes 500 KHz channelization and allows selection of 1 of 19 different channels in the 911.25 to 920.25 MHz sub-range of the ISM band.

The second sub-region (FCC_PART90_LOWBAND) utilizes 500 KHz channelization and allows selection of 1 of 2 different channels in the 902.75 to 903.25MHz sub-range of the ISM band.

4.2.3 Operating Mode

The `setup.operating_mode` variable controls the operational mode of the reader. There are four operational modes for the reader: active, standby, autonomous, and polled.

4.2.4 Protocols

The `setup.protocols` variable is used to configure which protocol(s) to utilize on the air interface. Currently, the reader can operate with ISOC, ISOB, FLEX, ISO10374, ISOB_80K, T21, PS111, and ASTMV6. Some protocols may require separate license.

4.2.5 Install Type

The `setup.install_type` variable indicates what type of application the reader is. This variable, along with the information in the region, sub_region, and tag_volume variables, provides a context for the reader to customize its detailed configuration settings to optimize performance.

For the 7204, the install_type defaults to **vehicle** which should be sufficient for most applications.

4.2.6 Tag Volume

The `setup.tag_volume` variable indicates the number of tags being presented to the reader at any one point in time. This variable, along with the information in the region, sub_region, and install_type variables, provides a context for the reader to customize its detailed configuration settings to optimize performance.

For the 7204, the tag_volume defaults to **1** which should be sufficient for most vehicle applications.

4.3 ANTENNA CONFIGURATION

The Neology Toll RFID Reader 7204 allows full antenna configuration. As a result, you can configure the following:

- Antenna Power
- Antenna Gain
- Cable Loss
- Attenuation

Neology Toll RFID Reader 7204 antennas should only be installed and maintained by Neology personnel or qualified electrical service personnel. Contact Neology authorized repair center for additional information on installing and configuring antennas and antenna parameters.

4.3.1 Setting the Antenna Power

The reader supports the ability to independently set the conducted power level for each antenna. It provides two methods for setting the conducted power for an antenna:

- User Set
- Reader Calculated

All antenna power settings are specified in ddBm (deci dBm or tenths of dBm). To convert from dBm to ddBm, multiply the dBm by 10. To convert from ddBm to dBm divide the ddBm by 10.

For example, to convert 29.3 dBm to ddBm, multiply the 29.3 by 10 = 293 ddBm. To convert 275 ddBm to dBm, divide the 275 by 10 = 27.5 dBm.

4.3.1.1 User Set Antenna Power

An antenna's conducted power can be set directly by setting the `antennas.X.conducted_power` variable (where the X is the number of the antenna (ie 1,2,3,4) to effect.

To set the conducted power for antenna number 1 to 27 dBm, use the following command:

```
>>> antennas.1.conducted_power = 270
ok
```

4.3.1.2 Reader Calculated Antenna Power

In order for the reader to calculate the conducted power for an antenna, you must specify the following:

- Antenna's gain

- Antenna's cable loss
- Attenuation to be added

With these inputs, the reader will calculate the appropriate power level for the configured regulatory region and sub-region. The attenuation level reduces the power level below the maximum allowed in the regulatory environment.

4.3.2 Specify the Antenna Gain

To specify the gain, the `antennas.1.advanced.gain_units` and `antennas.1.advanced.gain` configuration variables are used. The units supported are:

- dBdC
- dBiC
- dBd
- dBi

The gain specifications for a specific antenna should be available from the manufacturer and may be listed on the back of the antenna.

For example, to specify a gain for a linear polarized antenna of 13 dBi (130 in deci-dBi), use the commands:

```
>>> antennas.1.advanced.gain_units = dBi
ok

>>> antennas.1.advanced.gain = 130
ok
```

4.3.3 Specify the Cable Loss

To specify the antenna cable loss, set the `antennas.1.advanced.cable_loss` configuration variable. The loss specifications, in loss per unit length, for a specific type of cable should be available from the manufacturer.

For example, to set the cable loss for the entire length of cable to 2.5 dB (25 in deci-dB) use the command:

```
>>> antennas.1.advanced.cable_loss = 25
ok
```

4.3.4 Specify the Attenuation

To specify any attenuation to be added for the antenna you set the `antennas.1.advanced.attenuation` configuration variable. To set the attenuation to 1.5 dB (15 in deci-dB) use the command:

```
>>> antennas.1.advanced.attenuation = 15
ok
```

4.3.5 Reader Antenna Power Calculation

Once these variables are set, set the `antennas.1.conducted_power` to `0`. This will cause the reader to calculate the conducted power for the antenna.

```
>>> antennas.1.conducted_power = 0


ok
```

4.3.6 Sample Reader Antenna Power Calculation

As an example, consider the commands to cause the reader to calculate the conducted power with the following information:

Region = FCC and sub_region = FCC_PART90_DENSE which has a maximum allowable power of 30 Watts ERP (30 Watts = 44.8 dBm = 448 ddBm)

- Antenna gain units are specified in dBi
- Antenna gain is 130 ddBi
- Cable loss is 25 ddB
- 75 ddB of attenuation to be added

The command to configure these setting and the resulting reader calculated conducted power are as follows:

```
>>> antennas.1.conducted_power = 0
ok

>>> antennas.1.advanced.gain_units = dBi
ok

>>> antennas.1.advanced.gain = 130
ok
```

```
>>> antennas.1.advanced.cable_loss = 25
ok

>>> antennas.1.advanced.attenuation = 75
ok
```

Read the reader calculated conducted power with:

```
>>>
antennas.1.advanced.computed_conducted_power
ok 255
```

4.4 DIGITAL INPUT AND OUTPUT (DIO)

The 7204 is equipped with 0 to 3.3 Volt RS422/RS485 digital inputs/outputs (I/O) that provide four input signals and four output signals. The digital inputs can be used as general purpose inputs or to trigger the reader for tag reading. These inputs can be configured to provide an external read trigger from proximity sensors, photoswitches, or other devices.

The digital outputs can be used as general purpose outputs, to indicate tag reading activity, or to indicate the reader is transmitting (RF On). The outputs can also be configured to trigger conveyor gates or other access control and sorting devices.

Input	0 to 3.3 Vdc, RS422/RS485
Output	0 to 3.3 Vdc, RS422/RS485

The reader can control the state of the digital outputs and read the state of the digital inputs via writing and reading configuration variables. Additionally, asynchronous events are generated when the input or output states change. A user application could then use these events to trigger specific reader operations.

4.4.1 Reading Inputs

The current state of any specific digital input can be read by querying the **dio.in.X** variable, where X is in the range of 1 to 4 and corresponds to the four digital inputs. The result will be 1 (input is high) or 0 (input is low). For example:

```
>>> dio.in.1
ok 1

>>> dio.in.2
ok 0
```

You can also view the state of all inputs at once, by querying the `dio.in.all` variable. The result will be a hexadecimal number, representing the state of all four digital inputs. The least-significant-bit (LSB) corresponds to input 1 and the most-significant-bit (MSB) corresponds to input 4. In the following example, only input 2 is low:

```
>>> dio.in.all
ok 0xD
```

4.4.2 Reading and Writing Outputs

The current state of any specific digital output can be read by querying the `dio.out.X` variable, where X is in the range of 1 to 4 and corresponds to the four digital outputs. The state can be changed by setting the value of the `dio.out.X` variable. Setting the output value to a 0 connects the digital output to logic low, and setting it to a 1connects the digital output to logic high. The following example reads the current state of digital output 2 and then enables the output.

```
>>> dio.out.2
ok 1

>>> dio.out.2=0
ok
```

You can also view or set the state of all outputs at once, by querying or setting the `dio.out.all` variable. This variable contains a number, representing the state of all four digital outputs. The least-significant-bit (LSB) corresponds to output 1 and the most-significant-bit (MSB) corresponds to output 4.

In the following example, the state of the outputs is read and then all four outputs are changed (those that are on are switched off and vice versa):

```
>>> dio.out.all
ok 0xA

>>> dio.out.all=0x5
ok
```

4.4.3 Asynchronous notification of DIO changes

Whenever the state of digital inputs or digital outputs change, certain events are generated. Depending on the specific needs of your application, you may choose to register for some or all of these events.

- `event.dio.in.X value=Y` – This event indicates that digital input X has changed, and its new value is Y. X will be from 1 to 4, and Y will be either 0 or 1.

- `event.dio.out.X value=Y` – This event indicates that digital output X has changed, and its new value is Y. X will be from 1 to 4, and Y will be either 0 or 1.
- `event.dio.all in=X, out=Y` – This event is sent whenever any digital inputs or outputs change. It provides, in hexadecimal format, the state of all digital inputs (X) and all digital outputs (Y). The format of X and Y are the same as the format of the `dio.in.all` and `dio.out.all` variables.

For example, if all four digital inputs are high, and then digital input 3 goes low, the following two events are generated. In this case, digital outputs 2 and 4 are enabled (connected to ground):

```
event.dio.in.3 value=0


event.dio.all in=0x0b, out=0x05
```

4.4.4 Debouncing

Depending on the nature of the device connected to the digital inputs, some debouncing may be necessary. The variable `dio.debounce.X` sets a specific debounce time for each digital input. Input changes will only be reported after the new value has been asserted for greater than the amount of time specified by the debounce variable.

If very fast pulsed inputs are used, ensure the debounce time is not too low (less than the pulse width), or an input pulse may be missed. If a noisy switch contact is used as an input, ensure the debounce time is high enough to account for the noise on the contact.

The following example reads the debounce time for digital input 4, and then sets it to 100 ms.

```
>>> dio.debounce.4
ok 30


>>> dio.debounce.4=100
ok
```

4.4.5 Digital Input Alarm Generation

The 7204 can be configured to generate an alarm when a digital input is disconnected or sensor failure is detected. The alarm is triggered when the signal level on the digital input stays in the specified state longer than the specified alarm timeout. This behavior can be configured independently for each digital input.

The configuration variable `dio.in.alarm.logic_level.'N'` (where

'N' is 1,2,3,4) sets whether the alarm is coupled to a input logic level of 0 (low) or 1 (high).

The configuration variable `dio.in.alarm.timeout.'N'` (where 'N' is 1,2,3,4) sets the amount of time, in seconds, to wait for a signal state change. A value of 0 (default) disables alarm generation.

The digital input logic level along with the corresponding digital input pin timeout value is used to determine if an alarm (in the form of an event) should be generated. If a timeout value is set, the input pin is monitored.

If the input pin value does not change during the timeout period AND the input pin value matches the alarm logic level, the event `event.dio.in.alarm.timeout.n` (where n is the pin number) is generated. This alarm event generation can be helpful in alerting to the loss of digital inputs to the reader.

4.5 TAG OPERATIONS

This section describes operations that can be performed on tags or groups of tags. Many of these operations are applicable for all protocols, however, a few are protocol specific. Protocol-specific information can be found later in the next chapter.

All the functions in this section can take antenna as a parameter; if antenna is specified, the tag operation will be performed on that set of antennas. If antenna is not specified, the operation is attempted on all enabled antennas. Other arguments vary from function to function and are described in the following.

4.5.1 Read Tags

There are several functions which can be used to read various portions of a tag's set of information:

- `tag.read_id()` - obtains the EPC or tag ID from the first tag observed. The only argument to this function is the optional **antenna** parameter.

For the following functions, you can use the optional tag_id parameter to specify a specific tag to read (using the tag ID read with `tag.read_id()`). If the `tag_id` parameter is not specified, then the first observed tag is read.

- `tag.read_tid()` - reads the TID memory of a tag. If the memory is locked, the **pwd** argument can be used to specify the password.
- `tag.read_user_data()` - reads the User Data memory of a tag. If the memory is locked, the **pwd** argument can be used to specify the password.
- `tag.read_access_pwd()` - reads the tag's access password. If the memory is locked, the **pwd** argument can be used to specify the password.

- `tag.read_kill_pwd()` - reads the tag's kill password. If the memory is locked, the **pwd** argument can be used to specify the password.

The `tag.read()` function can be used in addition to or in place of the above `tag.read_*` commands. This function uses the `report` argument to specify which fields to read. This function provides an alternative method for reading multiple tag fields with one function.

In the following example, the tag ID, TID, and kill password are read from the tag using individual `tag.read_*` functions. Next, the same information is obtained with a single `tag.read` function.

```
>>> tag.read_id()

ok tag_id=0x3000214160C00400000A5937


>>>
tag.read_tid(tag_id=0x3000214160C00400000A593
7, antenna=1)

ok tid=0xE2001040


>>>
tag.read_kill_pwd(tag_id=0x3000214160C0040000
0A5937)

ok kill_pwd=0x00000000


>>> tag.read(report=tag_id tid kill_pwd)

ok tag_id=0x3000214160C00400000A5937,
kill_pwd=0x00000000,tid=0xE2001040
```

4.5.2 Write Tags

Similar functions are provided for writing tags. The following functions are all analogous to the read functions described earlier, but some use the `new_tag_id` parameter to specify the new EPC data to write. An additional optional parameter allows you to lock the newly written data.

`tag.write_id()`	Writes the tag's EPC or tag ID. In addition to the new_tag_id parameter, it also uses the optional lock_type parameter to lock or permalock the new data. The `tag_id` parameter (the old tag ID, used to specify a specific tag), and the optional pwd and antenna arguments are also used as described in the read functions.
`tag.write_tid()`	Writes the tag's TID. It uses the tid parameter to specify the new TID, as well as `lock_type`, `tag_id`, **pwd**, and **antenna** parameters.
`tag.write_user_data()`	Writes the tag's user data. It uses the `user_data` parameter to specify the new user data, as well as `lock_type`, `tag_id`, **pwd**, and antenna parameters.
`tag.write_access_pwd()`	Writes the tag's access password. It uses the `new_access_pwd` parameter to specify the new access password, as well as `lock_type`, `tag_id`, **pwd**, and **antenna** parameters.
`tag.write_kill_pwd()`	Writes the tag's kill password. It uses the kill_pwd parameter to specify the new kill password, as well as `lock_type`, `tag_id`, **pwd**, and **antenna** parameters.

As with the read functions, there is a single `tag.write()` command that can write multiple fields to a single tag. This command uses the `new_tag_id`, `kill_pwd`, `access_pwd`, **tid**, and `user_data` parameters. It also allows you to specify both the lock type (`lock_type`) and the data sections to lock (`lock_fields`) after the write is complete.

As with earlier functions, `tag_id` can be used to specify which tag to write, pwd can be supplied if required to write the tag, and antenna can be used to specify which antenna(s) to use.

In the following example, the tag ID and access password are written using individual commands. The data is then read back and written again using a single write command.

```
>>>
tag.write_id(new_tag_id=0x112233445566778899A
BCDEF)

ok


>>>
tag.write_access_pwd(new_access_pwd=0x9988776
6)
```

```
ok


>>> tag.read(report=access_pwd tag_id)

ok tag_id=0x112233445566778899ABCDEF,
access_pwd=0x99887766


>>>
tag.write(new_tag_id=0xAABBDDCCEEFF0011223344
55, access_pwd=0x12345678)

ok


>>> tag.read(report=access_pwd tag_id)

ok tag_id=0xAABBDDCCEEFF001122334455,
access_pwd=0x12345678
```

4.5.3 Kill Tags

To kill (permanently quiet) a tag, use the `tag.kill()` function. This function uses the following parameters:

- `tag_id` - Optional parameter specifying the tag id or EPC to kill.
- `kill_pwd` – Kill password.
- **antenna** - Optional parameter specifying which antenna(s) to use for the command. If antenna is not specified, all currently enabled antennas are used.

The following example kills a tag with kill password 0xaabbccdd. The EPC is not specified because only one tag is visible.

```
>>> tag.kill(kill_pwd=0xaabbccdd)

ok
```

4.5.4 Lock and Unlock Tags

As described previously, the reader can lock portions of the tag memory space while writing them. However, the reader can also change the lock state without first performing a write. In this case, the `tag.lock`* function are used. As described in the *Read Tags* and *Write Tags* sections, there are specific functions used to lock or unlock individual sections of a tag's memory space and a general function used to lock or unlock multiple sections.

The following functions are used to lock, unlock, perma-lock or perma-unlock a specific portion of a tag's memory space.

- `tag.lock_id()` - Changes lock state of the Tag ID or EPC of the tag.
- `tag.lock_tid()` - Change lock state of the TID of the tag.
- `tag.lock_user_data()` - Change lock state of the tag's user data.
- `tag.lock_access_pwd()` - Change lock state of the tag's access password.
- `tag.lock_kill_pwd()` - Change lock state of the tag's kill password.

Each function uses the following arguments:

- `lock_type` – Can be either **SECURED, UNSECURED, PERMA_LOCKED**, or **PERMA_UNSECURED**.
- `tag_id` - Optional parameter specifying the tag to lock.
- **pwd** - Optional parameter supplying the password required to change the tag's lock state.
- **antenna** - Optional parameter specifying which antenna(s) to use for the command. If antenna is not specified, all currently enabled antennas are used.

In addition to the specific tag lock functions, there is a general `tag.lock()` function that uses the `lock_fields` parameter. This parameter describes which fields will change lock state. This parameter is a string containing one or more of the following letters:

- `k` - kill password
- `a` - access password
- `e` - EPC
- `t` - TID
- `u` - user data

One additional function, `tag.unlock()`, is used to explicitly unlock portions of a tag's memory space. This function takes the same parameters as `tag.lock()` except that `lock_type` is not specified (assumes UNSECURED) and `unlock_fields` is used instead of `lock_fields`.

In the following example, the EPC and access password of a tag are locked using specific lock functions. The data is then unlocked using the general lock function. Note that the EPC is not specified because there is just one tag visible.

```
>>> tag.lock_id(lock_type=SECURED)
```

```
ok

>>> tag.lock_access_pwd(lock_type=SECURED)
ok

>>> tag.write_access_pwd(0x11223344)
error.tag.protocol.isoc.memory_locked info =
Memory Page = 0

>>> tag.lock(lock_type=UNSECURED,
lock_fields=ae)
ok
```

4.5.5 Tag Inventory / Asynchronous Events

After configuring all appropriate protocol and regulatory settings, the reader can begin singulating tags when `setup.operating_mode` is set to **active**. An `event.tag.report` event is generated each time a tag is singulated.

The format of the `event.tag.report` event is controlled by the `tag.reporting.report_fields` variable. This variable can contain the following fields displayed during each event.

- `tag_id` (Tag ID or EPC)
- `tid`
- `user_data`
- `user_data_offset` (memory offset where user_data report starts)
- `type` (protocol type)
- time
- antenna
- frequency
- rssi
- tx_power
- xpc
- gen2_timestamps
- prot_data

Data fields displayed in an `event.tag.report` event may include some or all fields requested by any of the following variables:

- `tag.reporting.report_field`
- `tag.reporting.arrive_field`
- `tag.reporting.depart_field`
- `tag.reporting.taglist_field`
- `tag.reporting.antenna_cross_fields`
- `tag.reporting.raw_arrive_fields`

Any software receiving this event should ignore those fields not needed for event handling.

The following example shows configuring the `event.tag.report` event to provide tag_id, type, and tid. It also shows the resulting `event.tag.report` event.

```
>>> tag.reporting.report_fields=tag_id type
tid
ok


event.tag.report
tag_id=0x3000214160C0040000A5937, type=ISOC,
tid=0xE2001040
```

Additional events are generated when a tag first is observed by the reader, and when it is no longer observed by the reader. These events are `event.tag.arrive`, `event.tag.raw_arrive`, and `event.tag.depart`.

4.5.5.1 Tag Arrival Event

The `event.tag.arrive` event can be configured by changing the `tag.reporting.arrive_fields` and `tag.reporting.raw_arrive_fields` variables. These variables can contain the following parameters:

- `tag_id` (Tag ID or EPC)
- tag_type_index (used with tag security features)
- tid
- user_data

- `user_data_offset` (memory offset where user_data report starts)
- **type** (protocol type)
- time
- label
- antenna
- rssi
- frequency
- audit_record (used with tag.db.create_entry_on_arrival)
- tid_authentic
- pw_authentic
- packet_counter
- writeback_state
- writeback_writes
- prot_data

The following example configures the `event.tag.arrive` event to provide `tag_id`, `type`, and `time`. It also shows the resulting `event.tag.arrive` event.

```
>>> tag.reporting.arrive_fields=tag_id type time
ok


event.tag.arrive tag_id=0x3000214160C0040000A5937,
type=ISOC,
first=2006-06-14T13:11:59.829
```

4.5.5.2 Tag Depart Event

The `event.tag.depart` event is controlled by two variables: `tag.reporting.depart_time` and `tag.reporting.depart_fields`.

- `tag.reporting.depart_time` controls when the reader considers the tag to have departed. If the reader does not observe the tag for the amount of time set in `tag.reporting.depart_time` (in ms), then it decides that the tag has departed and generates an `event.tag.depart` event.

- `tag.reporting.depart_fields` controls the contents of the event. This variable is a list that can include the following:
- `tag_id` (Tag ID or EPC)
- `tid`
- `user_data`

- `user_data_offset` (memory offset where user_data report starts)
- **type** (protocol type)
- time
- label
- antenna

- **repeat** (# of times the tag was observed since the arrival report).
- audit_record (used with tag.db.create_entry_on_arrival)
- packet_counter
- writeback_state
- writeback_writes
- prot_data label
- min_rssi
- max_rssi

Requesting `tid` or `user_data` will cause additional air interface overhead due to the additional transactions with the tag required to obtain that information.

The following example configures the `event.tag.depart` event to occur after a tag has not been observed for 1 second, and configures the event to contain the EPC, TID, antenna, and repeat count. The generated event is also shown.

```
>>> tag.reporting.depart_time=1000
ok
```

```
>>> tag.reporting.depart_fields=tag_id tid antenna
repeat
ok


event.tag.depart tag_id=0x3000214160C0040000A5937,
tid=0xE2001040, antenna=1, repeat=1646
```

4.5.6 Tag Database Operation

When the reader's `setup.operating_mode` variable is set to `active`, the reader will collect tag information in its tag database. You can then query the database periodically to observe tag information, instead of processing the asynchronous tag events. However, the events are still generated in this mode.

Two variables configure the tag database: `tag.reporting.taglist_fields` and `tag.db.max_count`.

- `tag.reporting.taglist_fields` specifies the fields reporting with each tag in the database. This variable is a list which can contain the following:
- tag_id (tag ID or EPC)
- tag_type_index (used with tag security features)
- tid
- user_data
- `user_data_offset` (memory offset where user_data report starts)
- type (protocol type)
- time (first and last times observed)
- antenna
- repeat (# of times the tag was observed)
- acknowledged (tag has been acknowledged with `tag.db.set_acknowledge`)
- audit_record (used with `tag.db.create_entry_on_arrival`)
- tid_authentic
- pw_authentic
- packet_counter
- crossed_antenna
- writeback_state
- writeback_writes

- prot_data
- audit_record
- min_rssi
- max_rssi

- `tag.db.max_count` sets up the tag database This variable controls how many tags can be stored in the database; its maximum value is 524,288.

The following functions perform basic database operations.

- `tag.db.get()` queries the database. This function returns all tags currently held in the database (all tags observed since the last purge of the database). All requested fields are returned for each tag.
- `tag.db.purge()` empties the database.
- `tag.db.get_and_purge()` read out the contents of the database and then empty it.
- `tag.db.scan_tags()` inventories all tags in the field for the specified number of milliseconds and then return the list of tags inventoried in that time.

In the following example, the tag database is configured to return EPC, time, and repeat count. The reader is put into active mode and the database is queried for tags.

```
>>> tag.reporting.taglist_fields=tag_id time
repeat
ok

>>> setup.operating_mode=active
ok

>>> tag.db.get()
ok

(tag_id=0x3000214160C00400000687E7,
first=2006-06-14T13:50:02.485, last=2006-06-
14T13:50:20.789, repeat=2012)
```

```
(tag_id=0x3000214160C00400000A5937,
first=2006-06-14T13:49:48.741, last=2006-06-
14T13:50:05.622, repeat=2175)
```

4.5.7 Tag Filtering

The 7204 supports the ability to filter tags. Filtering tags means to eliminate tags from being reported based on the conditions specified in the filter configuration variables.

4.5.7.1 Filter Configuration Variables

The reader supports eight filters. Each filter is specified by the following configuration variables:

- `tag.filter.n.name` **(STRING)**
- `tag.filter.n.enable` **(BOOL)**
- `tag.filter.n.pattern` **(ARRAY)**
- `tag.filter.n.mask` **(ARRAY)**
- `tag.filter.n.inclusive` **(BOOL)**

where:

`n` specifies the filter number (1 to 8). The **name** variable is provided to document the filter; it can be used to provide a name for each individual filter. The **enabled** variable is used to enable/disable the filter. The three remaining variables **pattern**, **mask** and **inclusive** specify the conditions for the filter.

A filter's conditions are based on bit-wise fields in the **mask** and **pattern** variables. Both the *pattern* and *mask* are specified as ARRAYS (a string of hex characters). In the *mask*, all bit positions where the value is important for the filtering are set to 1. The desired bit values to filter on are specified in the *pattern*.

A tag id matches the filter if the important bits (as specified by a **1** in the mask) of the pattern match the same bits in the tag id. Each of the filters is specified to be either **inclusive == true**, meaning that only tags matching the filter are reported, or **inclusive == false** (or exclusive), meaning that tags not matching the filter are reported. This can be described with the following algorithm:

```
if ((tag.filter.n.pattern  tag.filter.n.mask) ==
(tag_id  tag.filter.n.mask))
{
  // tag id matches the pattern with the mask.
  if (tag.filter.n.inclusive == true)
  {
  // report tag
  }
} else {
  // tag id does NOT matches the pattern with the
mask.
  if (tag.filter.n.inclusive == false)
  {
  // report tag
  }
}
```

4.5.7.2 Using Multiple Filters

Multiple filters can be specified. Each of the filters is specified to be either inclusive meaning that only tags matching the filter are reported, or exclusive, meaning that tags not matching the filter are reported. If multiple filters are used, a tag is reported only if the following two conditions hold:

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

if no inclusive patterns are defined, the first check is omitted.

4.5.7.3 Example Filter Usage

This section presents the filter setting to configure a reader to only report SGTIN-96 tags with a filter value indicating a Single Shipping/ Consumer Trade Item.

SGTIN-96 tags are identified with an 8-bit header value of 0011 0000 (binary) or 0x30 (hex). The next 3 bits of an SGTIN-96 tag are the filter value. The filter value is additional data that is used for fast filtering and pre-selection of basic logistics types. The filter values and their meaning are presented in the following table:

Type	Binary Value
All Others	000
Retail Consumer Trade Item	001
Standard Trade Item Grouping	010
Single Shipping/ Consumer Trade Item	011
Reserved	100
Reserved	101
Reserved	110
Reserved	111

SGTIN-96 tags with a filter value set to indicate a Single Shipping/ Consumer Trade Item would start with the bit pattern: 0011 0000 011.

To setup filter #1 on the reader for SGTIN-96 tags with a filter value indicating a Single Shipping/ Consumer Trade Item, use the following:

```
>>> tag.filter.1.name = SGTIN-96 SSCT
ok

>>> tag.filter.1.inclusive = true
ok

>>> tag.filter.1.mask = 0xFFE (only look at first
11 bits)
ok

>>> tag.filter.1.pattern = 0x306
ok

>>> tag.filter.1.enable = true
ok
```

5 EMBEDDED READER APPLICATIONS

5

EMBEDDED READER APPLICATIONS

The Neology Toll RFID Reader 7204 supports embedded user applications such as Python scripts. These applications can be developed on an external host and can be used to customize reader behavior to the customer's requirements, create new external interfaces, or maximize performance with time critical operations. This section describes how to load an embedded application.

5.1 DEVELOPING APPLICATION CODE

5.1.1 User Application Types Supported

The 7204 supports Python user applications. Python applications may be either in script format (**.py**) or compiled (**.pyc**).

The reader's Python version is 3.8.

5.1.2 Directories Available for User Applications

Users may use both **/tmp** (stored in RAM) and **/apps/bin** (stored in Flash) for their applications. Files and directories can be created and written to in these directories. Users may not "cd" to these directories, so it is advised that users use the fully qualified pathnames when referring to files in these directories.

User applications may use up to 16MB of the RAM file system (**/tmp**). Files stored in this file system are removed after a reboot. Other maintenance of this file system is up to the user.

User applications can use up to 64 MB of the Flash file system (**/apps/bin**). These files are preserved over a reboot. Due to the constraints for a flash file system, users should avoid adding files to the **/apps/bin** directory that are dynamically modified. Dynamic data can be maintained in `/tmp`.

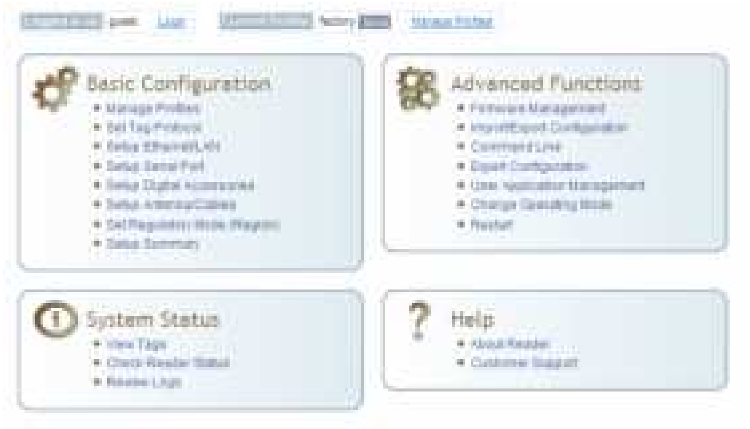
5.1.3 Registering for Reader Events

Events can also be registered by user applications. The Reader listens for Event connections on its local port 50008. User applications can connect to this port and will receive a channel id to use to register Events via the `reader.events.register()` command.

5.2 LOADING APPLICATION CODE ONTO THE READER

The 7204's embedded web interface provides a mechanism to transfer application files onto the reader. Perform the following:

1 From the RST main screen, press **Configure**. The 7204 Reader Configuration Tool (RCT) is displayed.



2 Under **Advanced Function**, select **User Application Management**.

3 Under **Application Transfer**, browse for the application file and select **Transfer File**.

4 The application is transferred to the reader.

5.3 LISTING APPLICATIONS

There are several sample applications shipped with the reader, including `hello.py` (Python script). These scripts and any other application can be listed using the `reader.apps.list()` function.

```
>>> reader.apps.list()
Ok apps = hello.py sample1.py dio_mon.py
```

5.4 RUNNING APPLICATIONS

After an application is loaded on the reader, you can start the application using the `reader.apps.start_python` function for Python scripts.

5.4.1 Python Script Example

The sample Python script `hello.py` is a simple script the prints `hello` and the argument passed to the script to standard output. To start the script, invoke it with the `reader.apps.start_python` command. In this example we'll pass the arguments test to the script.

```
>>> *reader.apps.start_python*
(filename=hello.py,args=test)
```

```
ok pid = 1963
```

The command returns the process id (pid) of the application. To see the status of the application, run the `reader.apps.list_running()` command.

```
>>> reader.apps.list_running()
ok running_apps = 1963:manual:ended:hello.py
test:
```

where output consist of:

```
PID:CONFIG:STATUS:COMMAND:ARGS
```

The output displays all the user commands that have been started on the reader. The column information displayed has the following meanings:

PID	Process id associated with the app.
CONFIG	**manual** - the application will not run at boot time.
	auto - the application will run at system boot.
STATUS	**starting** - the application is starting.
	running - the application is running.
	ended - the application has ended.
COMMAND	Name of application and parameters (if any) passed to application.

The output shows that `hello.py` has ended. To view the information passed to the standard output, run use `reader.apps.view_log` function, passing the pid associated with the application:

```
>>> reader.apps.view_log(1963)
ok


Hello! args passed: test
```

To set this application to run at system boot, pass the **autostart** parameter to the `reader.apps.start_python` command:

```
>>> reader.apps.start_python
(filename=hello.py,args=test, autostart=true)
ok pid = 2018
```

Running `reader.apps.list_running` shows that the application is configured as `auto`, which means it will run again at system boot time:

```
ok running_apps = 1963:manual:ended:hello.py
test:
    2018:auto:ended:hello.py test:
```

To remove an application, run the `reader.apps.stop` function and pass the pid associated with the application:

```
    >>> reader.apps.stop(1963)
    ok


    >>> reader.apps.list_running
    ok running_apps = 2018:auto:ended:hello.py
    test:
```

5.5 VIEWING APPLICATION CREATED FILES ON THE READER

Files created by user applications in the **/apps/bin** directory can be viewed from the reader's User Application Management web page.

Select the file to be viewed using the dropdown box under the **Applications available on the reader** section and press the **View** button to view the file. These files must be created with permissions of 444 or greater to view from the web. Alternately, files may be offloaded from the reader via the **scp** utility that may be executed from the user application.

5.6 READER FIREWALL CONSIDERATIONS

The Neology Toll RFID Reader 7204 utilizes a firewall which blocks most incoming network traffic. If your embedded application needs to be able to receive incoming network traffic from external devices, your application should use ports 50010 through 50015 (both TCP and UDP). Incoming traffic on all other ports will be blocked.

6 TAG PROTOCOL SUPPORT

6 TAG PROTOCOL SUPPORT

6.1 ISO 18000-63 (ISO-C) PROTOCOL SUPPORT

In order to enable the ISO-C protocol, the `setup.protocols` variable must contain the `isoc` parameter. In general, `setup.install_type` and `setup.tag_volume` will automatically configure the ISO-C parameters. This section describes how to configure specific ISO-C parameters for more precise control.

6.1.1 Protocol Configuration

The `modem.protocol.isoc.control` namespace configures various protocol level parameters. The following variables are available within that namespace:

cmd_retries	Controls how many times each ISO-C command is attempted quietly before returning an error message
display_tag_ crc	If true, all tag reports include the CRC bits in the tag_id. If false, only actual EPC bits are included
inventory_both_targets	If true, each inventory round toggles tag's inventory state between A and B. If false, each round only inventories tags in the initial query target state to the opposite state.
number_slots_q	At beginning of inventory round, this is value of Q (log2 [number of slots])
max_incr_slots_q	As inventory round runs, Q is adjusted up or down dynamically; this variable controls how much larger Q can grow. Ex: If number_slots_q is 4 and max_incr_slots_q is 2, then Q is capped at 6.
mem_bank_for_selection	When select commands are sent, controls the memory bank used (see Filtering section).
session_id	Controls which ISO-C session (S0 through S3) is used for each inventory round.
use_block_write	If true, block writes used (not supported by all tags). If false, 16 bit writes used (supported by all tags).
select_cmd_period	If non-zero, specifies how many inventory rounds are performed prior to issuing the select command.
Query_target	Sets initial inventoried flag state used in initial query command.

6.1.2 Physical Layer Configuration

Various physical layer parameters can be configured under the `modem.protocol.isoc.physical` namespace. Under this namespace are the following variables:

`return_link_freq`	Read only. Controls link frequency tags use to respond to reader. Values: LF80, LF240, LF320, LF640 where number indicates frequency in thousands.
`Tari`	Controls length of tari (reference time for data-0). It is read-only and has one of the following values: TARI_06_25, TARI_08_33, TARI_12_50, or TARI_25_00. The number indicates the tari in microseconds; for example, TARI_12_50 indicates a 12.5 µs tari.
`data_1_length`	Controls the relative length of a data-1 symbol, as compared to data-0. Valid choices for this read-only variable are: D1_LEN_15 (each data-1 is 1.5 times the length of a data-0) or D1_LEN_20 (each data-1 is 2.0 times the length of a data-0).
`rt_modulation`	Controls modulation type used in the reader-to-tag transmission path. It is read-only and has the following values: rt_mod_dsb (DSB or double-sideband modulation) or rt_mod_pr (PR-ASK or phase-reversal amplitude shift keying modulation)
`tr_encoding`	Controls the encoding used on the tag-to-reader transmission path. It is read-only and has the following values: tr_enc_fm0 (FM0 encoding) or tr_enc_miller_X where X is 2, 4, or 8 (Miller-2, Miller-4, or Miller-8 encoding)

As described previously, `return_link_freq`, **tari**, and `data_1_length` are all read-only. In order to configure these variables, the `modem.protocol.isoc.physical.set()` function is used to set all three variables simultaneously:

```
modem.protocol.isoc.physical.set(tari=tari_10
6_25, return_link_freq=LF640,
data_1_length=d1_len_20)
```

6.2 FILTERING

The reader can send additional ISO-C Select commands to filter the tag population prior to an inventory round. To enable filtering in the reader, the `modem.protocol.isoc.filtering.enable` variable must be set to **true**.

Once set, select commands are sent if any filters are enabled, regardless of the `modem.protocol.isoc.control.select_cmd_enable` variable's setting.

Each filter uses a `modem.protocol.isoc.filter.X` namespace, where `X` is `1` to `8` for the 8 filters. The following variables exist in each namespace:

Enabled	If true, this filter is enabled.
Action	What action to take with this filter. Choices include: ASSERT_DEASSERT, ASSERT_NOTHING, NOTHING_DEASSERT, NEGATE_NOTHING, DEASSERT_ASSERT, DEASSERT_NOTHING, NOTHING_ASSERT, NOTHING_NEGATE. Each choice corresponds to a Select Action described in the ISO-C specification. The first part of each choice describes actions taken with tags that match the filter (Assert inventoried or SL, deassert it, negate it, or do nothing). The second part describes actions taken with tags that do not match.
Length	Length, in bits, of this filter.
Offset	Bit location where the mask comparison will begin.
Mask	This mask is compared against the tag's memory to determine if the tag matches. The first bit of this mask is compared to the offset bit of the tag's memory, and the mask should be length bits long.
Mem_bank	The memory bank used for filtering. Choices include: MEMBANK_EPC, MEMBANK_TID, MEMBANK_USER, NOT_USED. If NOT_USED is selected, the variable `modem.protocol.isoc.control.mem_bank_for_selection`controls which memory bank is used for this filter.
session	Session target for ISO-C mask filter. This is session inventoried flag or SL flag when a tag matches the filter. If set to NOT_USED, then `modem.protocol.isoc.filtering.use_session` and `modem.protocol.isoc.control.session_id` will control whether a session or select flag is used for this filter. Choices include: `S0 S1 S2 S3 SL NOT_USED`

7 READER NAMESPACE

7

READER NAMESPACE

7.1 READER

reader.check_status

Checks and returns reader status.

Class	FUNCTION		
Parameters	**cpu_only**	:	BOOL
Response	STRINGLIST		
Permissions	guest	x	
	admin	x	

This function causes the reader to perform internal checks and returns the status of these checks. The information returned includes:

- `reader_uptime` - Number of seconds the reader has been running for,
- `in_use_memory` - Amount of memory in use within the reader,
- `free_memory` - Amount free memory available within the reader,
- `cpu_load_user` - CPU load for user in percent (see `reader.cpu_monitor()`),
- `cpu_load_system` - CPU load for system in percent, (see `reader.cpu_monitor()`)
- `modem_alive` - Indication the modem is up and running,
- `modem_uptime` - Number of seconds the modem has been running for,
- `antenna_status` - Status of the antenna(s),
- `tx_interlock` - Condition of regulatory interlock status,
- `synth_locked` - Indication of the synthesizer lock status,
- `ps_fault` - Power supply fault status
- filesystem:/ - Amount of flash in use by the reader firmware filesystem
- filesystem:/var/volatile - Amount of RAMFS in use by the reader
- filesystem:/apps - Amount of flash in use by the reader applications If the `cpu_only` parameter is specified and set to `TRUE`, only the cpu based statistics will be reported back (`in_use_memory`, `free_memory`, `cpu_load\*`).

The following example gets the status of a reader:

```
>>> reader.check_status()
ok
reader_uptime = 178,
in_use_memory = 43716608,
free_memory = 19787776,
in_use_flash = 716608,
cpu_load_user = 4,
cpu_load_system = 5,
modem_alive = true,
modem_uptime = 160,
antenna_status = ok,
tx_interlock = false,
synth_locked = true,
ps_fault = false
```

reader.cpu_monitor

Obtain CPU user and system load utilization

Class	`FUNCTION`	
Parameters		
Response	`STRINGLIST`	
Permissions	guest	–
	admin	x

Obtain CPU user (application) and system load utilization percentage. The result reflects the load between the current and previous execution of this function. This is a light weight method to get a snapshot of processor loading. The following example obtain processor loading since last invocation.

```
>>> reader.cpu_monitor()
ok
cpu load user 0.45%,
cpu load system 0.79%
```

7

READER NAMESPACE

reader.create_csd

Creates customer support data (csd)

Class	FUNCTION	
Parameters		
Response	BOOL	
Permissions	guest	x
	admin	x

This function creates customer support data to be obtained via the web Customer Support page or via scp (secure copy). It creates an encrypted file in the tmp directory called "/tmp/`csd.sir` that can be sent to support.

reader.flash_led

Flash the reader LEDs

Class	FUNCTION		
Parameters	**led**	:	INT
	time	:	INT
Response	BOOL		
Permissions	guest	x	
	admin	x	

This function flashes the reader LEDs. The led parameter is a bitmask for led and color where 1=power led green, 2=power led red 4=actv led green, 8=actv led red 16=env/user led green, 32=env/user led red 64=stat led green, 128=stat led red. The time parameter specifies the on-off duty cycle and must be specified in milli-seconds.

the minimum allowed on-off time is 50 milli-seconds. The LEDs will flash continuously until canceled. To cancel all flashing, set time=0, with any led parameter value. If no parameters are entered, all LEDs will flash for 5 seconds.

modem actively sets the actv led, when executing flashing command, you may not see the length of the pulse as you set. You may see a brief flash instead.

The following example continuously flashes both the power green and env/user green LEDs for 1 second on, 1 sec off.

```
>>> reader.flash_led(led=17, time=1000)
ok
```

reader.is_alive

Verifies if reader is operational.

Class	FUNCTION	
Parameters		
Response	BOOL	
Permissions	guest	x
	admin	x

This function verifies if the reader is operational.

The following example shows how to use this command to verify the reader is alive and responsive.

```
>>> reader.is_alive ()
ok
```

reader.login

Allows reader login.

Class	FUNCTION		
Parameters	**login**	:	STRING
	pwd	:	STRING
Response	BOOL		
Permissions	guest	x	
	admin	x	

This function allows users to login to the reader.

The following example shows an administrator login.

```
>>> reader.login (login = admin, pwd =
1Hjg7df45)
```

```
ok
```

reader.logout

Allows reader logout.

Class	`FUNCTION`	
Parameters		
Response	`BOOL`	
Permissions	guest	`x`
	admin	`x`

This function allows users to logout of the reader. By default the login is changed to guest.

The following example shows a typical user logout:

```
>>> reader.logout()
ok
```

reader.reboot

Reboots the reader.

Class	`FUNCTION`	
Parameters		
Response	``	
Permissions	guest	–
	admin	`x`

This function reboots the reader.

The following example forces the reader to reboot (function does not return any response):

```
>>> reader.reboot()
```

7 READER NAMESPACE

reader.set_pwd

Changes reader passwords.

Class	`FUNCTION`		
Parameters	**login**	:	`STRING`
	pwd	:	`STRING`
	new_pwd	:	`STRING`
Response	`BOOL`		
Permissions	guest	`x`	
	admin	`x`	

This function changes the reader password to the password specified in the function parameter. The old and new password must be specified. You can, however, set a lower level login password without specifying the old password. The password can have a maximum length of 64 characters. Any printable alphanumeric character is allowed in the password, except for the following: (open parenthesis) close parenthesis , comma " double quote ' single quote

The following example shows an administrator password change from 1Hjg7df45 to Tat1001:

```
>>> reader.set_pwd (login = admin, pwd =
1Hjg7df45,
new_pwd = Tat1001)
ok
```

reader.timestamp_all_events

Timestamp all events.

Class	`VAR`	
Type	`bool`	
	default value: `FALSE`	
Permissions	guest	`r`
	admin	`rw`

When `true`, all events that do not have finer-granularity timestamp control will contain a timestamp. Certain events such as `event.tag.report` or `event.tag.arrive` have specific variables controlling whether or not they contain a timestamp, and are not affected by this variable.

reader.view_log

Displays a log file.

Class	FUNCTION		
Parameters	**log_file**	:	ENUM DEFINITIONS.ENUM.READER.LOG_FILES
	stream	:	BOOL
Response	COMPOUNDLIST		
Permissions	guest	x	
	admin	x	

This function displays the selected log file. The "stream" variable defaults to `TRUE`. When `TRUE`, `reader.view_log()` will return the log file as part of the cmd/resp channel stream. If `FALSE`, `reader.view_log()` will create the file on the reader and return the location of that file on the reader. The file can then be retrieved via a file copy mechanism like SFTP or SCP.

```
Example with stream set to TRUE :
```

The following example displays the reader error log

```
>>> reader.view_log(READER_ERROR_LOG)
Example with stream set to FALSE :
```

The following example displays the reader error log

```
>>> reader.view_log(READER_ERROR_LOG, FALSE)
ok /tmp/view_log.reader_error_log
```

reader.who_am_i

Reports the current login level.

Class	FUNCTION	
Parameters		
Response	STRING	
Permissions	guest	x
	admin	x

This function reports the user's login level.

The following example returns a response indicating the user is a guest.

```
>>> reader.who_am_i()
ok guest
```

7.2 READER.APPS

reader.apps.delete

Deletes a user application from the reader.

Class	FUNCTION		
Parameters	**filename**	:	STRING
Response	BOOL		
Permissions	guest	–	
	admin	x	

This function deletes a user application that is on the reader.

The following example deletes an application called `test.py`

```
>>> reader.apps.delete(filename=test.py)
ok
```

reader.apps.export_package

Exports all applications startup scripts.

Class	FUNCTION	
Parameters		
Response	BOOL	
Permissions	guest	–
	admin	x

This function exports all applications on the reader as well as their autostart settings and startup arguments to a file called /tmp/`apps_package.neoa;` this file can then be retrieved via scp. This functionality is also available via RCT (reader web pages).

reader.apps.import_package

Imports applications startup scripts.

Class	FUNCTION		
Parameters	**filename**	:	STRING
Response	BOOL		
Permissions	guest	–	
	admin	x	

This function imports the applications startup information contained in the supplied file.

this will overwrite any user applications with the same filenames that already exist on the reader. This functionality is also available via RCT (reader web pages).

reader.apps.list

Returns list of all user applications loaded on the reader.

Class	FUNCTION	
Parameters		
Response	STRING	
Permissions	guest	x
	admin	x

This function returns a list of all user applications loaded on the reader. Applications can be started with the `reader.apps.start_python` and can be stopped with the `reader.apps.stop()` function. The list of currently running applications can be obtained with the `reader.apps.list_running()` function.

The following example lists all user applications loaded on the reader:

```
>>> reader.apps.list()
ok apps = dio.py test.py
```

reader.apps.list_running

Returns list of all user applications that are running on the reader.

Class	`FUNCTION`	
Parameters		
Response	`STRING`	
Permissions	guest	`x`
	admin	`x`

This function returns a list of all user applications running on the reader. Running applications are stopped with the `reader.apps.stop()` function.

The following example lists all user applications running on the reader:

```
>>> reader.apps.list_running()
ok running_apps = 8537:auto:running:dio.py 1
;
```

reader.apps.start_python

Executes a user python script.

Class	`FUNCTION`		
Parameters	**filename**	:	`STRING`
	args	:	`STRING`
	autostart	:	`BOOL`
Response	`STRING`		
Permissions	guest	–	
	admin	`x`	

This function executes the user python script specified by the filename parameter (required). Runtime arguments (optional) can be passed to the script. If the autostart parameter is set to `true`, the scripts executes immediately and is set to run at the next system boot. If the auto start parameter is set to `false` (the default), the script will run immediately, but will not run again at the next system boot. The process ID (PID) associated with the function is returned. The status of the running apps can be displayed with the `reader.apps.list_running()` command. If the script sends information to stdout or stderr, it can be viewed with the `reader.apps.view_log()` command, passing the PID of the script.

The following example starts a python script named `dio.py` passing it the argument "1" and making the script autostartable:

```
>>> reader.apps.start_python(dio.py, "1",
true)
ok pid = 8537
```

reader.apps.stop

Stops a running user script or java application.

Class	FUNCTION		
Parameters	**pid**	:	INT
Response	BOOL		
Permissions	guest	–	
	admin	x	

This function stops the user script or java application previously started with the `reader.apps.start_python()` or `reader.apps.start_java()` function.

The following example stops a previously started application with a PID of 8537:

```
>>> reader.apps.stop(8537)
ok
```

reader.apps.stop_all

Stops all running user applications

Class	FUNCTION	
Parameters		
Response	BOOL	
Permissions	guest	–
	admin	x

This function stops all user scripts or java applications previously started with the `reader.apps.start_python()` or `reader.apps.start_java()` function.

The following example stops all previously started application

```
>>> reader.apps.stop_all()
ok
```

reader.apps.view_log

View an application log file.

Class	FUNCTION		
Parameters	**pid**	:	INT
Response	STRING		
Permissions	guest	–	
	admin	x	

This function displays an application log file. If the file is greater than 10K bytes in size, the last 10K bytes of the file are displayed.

The following example views an application's log file:

```
>>> reader.apps.view_log(pid = 8537)
ok
dio 1 went high, starting scan for tags
dio 1 went low, stop scan for tags
scan found 196 tags
```

7.3 READER.EVENTS

reader.events.bind

Sets the event channel for future calls to `reader.events.register` function.

Class	FUNCTION		
Parameters	**id**	:	INT
Response	BOOL		
Permissions	guest	x	
	admin	x	

This function sets the event channel for future calls to the `reader.events.register()` function. Once connected to the event channel, a channel ID is provided. The bind function uses this channel ID to set the event channel so it does not need to be passed in on future calls to `reader.events.register.`

The following example binds future calls to `reader.events.register()` to channel ID 14 and registers for the `event.tag.report` on event channel 14:

```
>>> reader.events.bind(id = 14)
ok
>>> reader.events.register(name =
event.tag.report)
ok
```

reader.events.buffer

Buffer events for higher throughput

Class	VAR	
Type	bool	
	default value: TRUE	
Permissions	guest	r
	admin	rw

When `true`, all events go through a buffer to improve throughput. This may have the side effect, in some apps, to increase latency (though not likely if normal TCP is used).

reader.events.list_registered

Lists the channel ids for each event that has been registered

Class	FUNCTION	
Parameters	**id** :	INT
Response	COMPOUNDLIST	
Permissions	guest	–
	admin	x

This command lists information indicating what event channels have registered for what events. An input for an event channel id can be provided so that only events registered on that specific event channel will be displayed.

The following example shows using the `reader.events.list_registered` command to show what event channels are registered for what events.

```
-->reader.events.list_registered()
ok
( event = event.tag.report, fds = 15)
( event = event.tag.arrive, fds = 16)
( event = event.tag.depart, fds = 16)
( event = event.tag.scan_tags_complete, fds =
17)
( event = event.status, fds = 17)
If an input for event channel id is provided,
it will filter the
output as shown below.
-->reader.events.list_registered(16)
ok
( event = event.tag.arrive, fds = 16)
( event = event.tag.depart, fds = 16)
```

reader.events.query_bind

Query the event channel ID for the command session.

Class	FUNCTION	
Parameters		
Response	STRING	
Permissions	guest	x
	admin	x

This function returns the event channel ID associated with the current command session.

The following example returns the event channel ID associated with the current command channel.

```
>>> reader.events.query_bind()
```

```
ok
```

reader.events.register

Registers events over event channel

Class	FUNCTION		
Parameters	**id**	:	INT
	name	:	STRING
Response	BOOL		
Permissions	guest	x	
	admin	x	

This function registers to have events delivered over an event channel. Once connected to an event channel, a channel id is provided. This channel id, along with event name, are passed to this function to register for the specified event.

```
The first example registers for
'event.tag.report' on event

channel 14.

The second and third examples register for
'event.tag.arrive'

and 'event.tag.depart' on event channel 16.

The fourth example registers for all
'event.tag.*' events on

event channel 17.

>>> reader.events.register(14,
event.tag.report)

ok

>>> reader.events.register(16,
event.tag.arrive)

ok

>>> reader.events.register(16,
event.tag.depart)

ok

>>> reader.events.register(17, event.tag)

ok
```

reader.events.trigger

Distributes event to all registered event channels.

Class	FUNCTION		
Parameters	**name**	:	STRING
Response	BOOL		
Permissions	guest	x	
	admin	x	

This function is used by a client to distribute an event to all registered event channels.

The following example triggers `event.app.start_business_process` (sends it out to all registered event channels):

```
>>> reader.events.trigger(name =
"event.app.start_business_process ap_id =
23")
ok
```

reader.events.unregister

Unregisters events over a event channel

Class	FUNCTION		
Parameters	**id**	:	INT
	name	:	STRING
Response	BOOL		
Permissions	guest	x	
	admin	x	

This function unregisters events on a event channel.

The following example un-registers for the `event.tag.depart` on event channel 16.

```
>>> reader.events.unregister(id=16,
name=event.tag.depart)
ok
```

7.4 READER.FIRMWARE

reader.firmware.rollback

Rolls back the reader firmware.

Class	FUNCTION	
Parameters		
Response	BOOL	
Permissions	guest	–
	admin	x

This function rolls back the reader firmware to the previous version.

The following example rolls back the reader firmware to the previous version.

```
>>> reader.firmware.rollback()
ok
```

Following the conclusion of the rollback, reboot the reader by executing the reboot command.

```
>>> reader.reboot()
ok
```

reader.firmware.upgrade

Upgrades reader firmware.

Class	FUNCTION	
Parameters	**filename**	: STRING
Response	BOOL	
Permissions	guest	–
	admin	x

This function upgrades the reader firmware with the file specified.

The following example upgrades the reader to the firmware contained in the `latest.neo1` file.

```
>>> reader.firmware.upgrade
(filename=latest.neo1)
ok
```

Following the conclusion of the upgrade, reboot the reader by executing the reboot command.

```
>>> reader.reboot()
ok
```

7.5 READER.LICENSE

reader.license.import

Imports license file to reader.

Class	FUNCTION		
Parameters	**filename**	:	STRING
Response	BOOL		
Permissions	guest	–	
	admin	x	

This function imports a license file to the reader. It will check the license file for proper match to the reader MAC address and proper checksums. If the licenses are appropriate to the specific reader doing the importation and they haven't been used previously, they will be accepted and the corresponding licensable feature will be enabled.

The following example imports reader license file.

```
>>>
reader.license.import(filename=license.txt)
ok
```

reader.license.list_order_ids

List license order ids

Class	FUNCTION	
Parameters		
Response	BOOL	
Permissions	guest	–
	admin	x

Lists the order ids currently stored on reader for licenses imported in past.

reader.license.status

Responds with enable/disable status of reader licenses.

Class	`FUNCTION`	
Parameters		
Response	`STRING`	
Permissions	guest	–
	admin	x

This function lists the enable/disable status of the imported reader licenses (and those enabled by default).

7.6 READER.PROFILE

reader.profile.active

Holds the active profile for the reader.

Class	`VAR`	
Type	`string`	
	default value: `"factory"`	
Permissions	guest	r
	admin	r

This variable holds the name of the profile that is the current active profile in the reader.

The following example shows the use of this command to get the name of the active profile in use on the reader.

```
>>> reader.profile.active
ok isoc_portal
```

reader.profile.delete

Deletes the specified profile.

Class	FUNCTION		
Parameters	**filename**	:	STRING
Response	BOOL		
Permissions	guest	–	
	admin	x	

This function deletes the specified profile from the reader. This is a destructive operation. Once a profile is deleted, it cannot be recovered.

The following example deletes a `test_profile` from the reader. This is a destructive operation. Once the `test_profile` profile is deleted, it cannot be recovered.

```
>>> reader.profile.delete(test_profile)
ok
```

reader.profile.list

Returns list of all previously saved profiles.

Class	FUNCTION	
Parameters		
Response	STRING	
Permissions	guest	–
	admin	x

This function returns a list of all previously saved reader profiles.

The following example lists all profiles on the reader.

```
>>> reader.profile.list()
ok profiles = isoc_portal isoc_conveyor
isob_portal
test_profile
```

reader.profile.load

Loads specified reader profile.

Class	`FUNCTION`		
Parameters	**filename**	:	`STRING`
Response	`BOOL`		
Permissions	guest	–	
	admin	x	

This function loads the specified profile of reader settings. Once a profile is loaded, it is the active profile (see `reader.profile.active`) and it is the profile loaded at reader startup.

The following example loads the reader profile`isoc_portal`. Once `isoc_portal` is loaded, it is the active profile (see `reader.profile.active`) and it is the profile loaded at reader startup.

```
>>> reader.profile.load(isoc_portal)
ok
```

reader.profile.reset_factory_default

Reset reader profile to the factory default.

Class	`FUNCTION`	
Parameters		
Response	`BOOL`	
Permissions	guest	–
	admin	x

This function resets the reader profile to the factory default profile.

The following example resets the reader to its factory defaults.

```
>>> reader.profile.reset_factory_defaults()
ok
```

reader.profile.save

Saves current reader configuration to specified profile.

Class	FUNCTION		
Parameters	**filename**	:	STRING
Response	BOOL		
Permissions	guest	–	
	admin	x	

This function saves the reader's current configuration to the specified profile. Once a profile is saved, it is the active profile (see `reader.profile.active`) and is the profile loaded at reader startup.

Profile names must consist of the characters A - Z, a - z, 0 - 9, '-' or '_' and must be between 1 and 32 characters in length. The reader can store up to 8 different profiles.

The following example saves the reader's current settings to a profile named `isoc_conveyor`. Once the `isoc_conveyor` profile is saved, it is the active profile (see `reader.profile.active`) and is the profile loaded at reader startup.

```
>>> reader.profile.save(isoc_conveyor)
ok
```

reader.profile.show_running_config

Show the current running configuration of the reader.

Class	FUNCTION
Parameters	
Response	STRING
Permissions	guest –
	admin x

This function shows the current running configuation parameters of the reader.

The following shows the readers current running configuration.

```
>>> reader.profile.show_running_config()
ok setup.install_type portal
setup.operating_mode standby
setup.protocols isoc setup.region fcc . . .
```

7.7 READER.SERVICE

reader.service.check

Checks and returns status of a system service.

Class	`FUNCTION`
Parameters	**program** : `ENUM DEFINITIONS.ENUM.SERVICE_NAME`
Response	`STRINGLIST`
Permissions	guest `x`
	admin `x`

This function checks the status of a system service. The information returned could be one of the following:

- `<program>` is alive, pid=`<pid>`
- `<program>` is not running
- `<program>` is dead, last know pid=`<pid>`

The following example gets the status of the discovery service:

```
>>> reader.service.check(discovery)
ok
discovery is alive, pid=14573
```

reader.service.start

Starts a system service.

Class	`FUNCTION`
Parameters	**program** : `ENUM DEFINITIONS.ENUM.SERVICE_NAME`
	autostart : `BOOL`
Response	`BOOL`
Permissions	guest `–`
	admin `x`

This function starts a system service.

The following example starts the discovery service:

```
>>> reader.service.start(discovery)
ok
```

reader.service.stop

Stops a system service.

Class	FUNCTION		
Parameters	**program**	:	ENUM DEFINITIONS.ENUM.SERVICE_NAME
	autostart	:	BOOL
Response	BOOL		
Permissions	guest	–	
	admin	x	

This function stops a system service.

The following example stops the discovery service:

```
>>> reader.service.stop(discovery)
ok
```

8 SETUP NAMESPACE

8 SETUP NAMESPACE

8.1 SETUP

setup.default_login_level

Sets default login level for cmd sessions.

Class	`VAR`	
Type	`enum`	
	default value: `GUEST`	
Permissions	guest	`r`
	admin	`rw`

This variable sets the login level assigned to all new cmd sessions.

The following example sets the default login level for the reader:

```
>>> setup.default_login_level = admin
ok
```

setup.install_type

Configures reader installation type.

Class	`VAR`	
Type	`enum`	
	default value: `VEHICLE_TOLLING`	
Permissions	guest	`r`
	admin	`rw`

This variable configures the reader for the type of installation. Based on the `install_type`, Distance, Power and Modulation Depth are set in the reader.

The following example configures the reader for portal operation:

```
>>> setup.install_type = vehicle_tolling
ok
```

8 SETUP NAMESPACE

setup.operating_mode

Configures reader operating mode.

Class	`VAR`	
Type	`enum`	
	default value: `Standby`	
Permissions	guest	`r`
	admin	`rw`

This variable configures the reader to read continuously (Active mode), or to wait for individual tag read commands (Standby mode).

The following example configures the reader to run continuously:

```
>>> setup.operating_mode = active
ok
```

setup.protocols

Configures tag protocols.

Class	`VAR`	
Type	`enumlist`	
	default value: `ISOC`	
Permissions	guest	`r`
	admin	`rw`

This variable configures the list of tag protocols supported by the reader.

The following example configures the reader to inventory ISOB and ISOC tags:

```
>>> setup.protocols = ISOB ISOC
ok
```

8 SETUP NAMESPACE

setup.region

Configures reader for geographic region.

Class	`VAR`	
Type	`enum`	
	default value: `UNSELECTED`	
Permissions	guest	`r`
	admin	`rw`

This variable configures the reader for a specific geographic region.

The following example configures the reader for regions covered by ETSI regulations:

```
>>> setup.region = ETSI
ok
```

setup.sub_region

Configures reader for a sub-region within geographic region.

Class	`VAR`	
Type	`enum`	
	default value: `UNSELECTED`	
Permissions	guest	`r`
	admin	`rw`

This variable configures the reader for a sub-region within the major geographic region.

The following example configures the reader for a sub-region covered by `EN302208_dense` regulations:

```
>>> setup.sub_region = en302208_dense
ok
```

setup.tag_volume

Configures reader for estimated number of tags in field.

Class	`VAR`	
Type	`enum`	
	default value: `1`	
Permissions	guest	`r`
	admin	`rw`

This variable configures the reader for an estimated number of tags in the field.

The following example configures the reader for 200 tags in the field:

```
>>> setup.tag_volume = 128_256
ok
```

setup.valid_isoc_phy_list

Sets which ISOC physical settings are valid.

Class	`VAR`	
Type	`enumlist`	
	default value: `6d_640_m_2 8p_640_m_2 12d_320_m_2 25p_240_m_4 25p_80_m_1 25d_240_m_2`	
Permissions	guest	`–`
	admin	`r`

This variable indicates which ISOC physical settings are valid and/or available for this reader.

8 SETUP NAMESPACE

setup.valid_protocols

Sets which protocols are valid/available.

Class	`VAR`	
Type	`enumlist`	
	default value: `ISOC`	
Permissions	guest	–
	admin	r

This variable indicates which protocols are valid and/or available for this reader.

setup.valid_regions

List the regions the reader can be configured for.

Class	`VAR`	
Type	`enumlist`	
	default value: `UNSELECTED`	
Permissions	guest	–
	admin	r

This variable returns all regions for which the reader is currently configured.

The following example indicates the reader is currently supporting the ETSI region:

```
>>> setup.valid_regions
ok fcc etsi
```

setup.valid_sub_regions

List the `sub_regions` the reader can be configured for.

Class	`VAR`	
Type	`enumlist`	
	default value: `UNSELECTED`	
Permissions	guest	–
	admin	r

This variable lists all `sub_regions` for which the reader is currently configured.

The following example returns the currently supported sub-regions:

```
>>> setup.valid_sub_regions
ok fcc_a fcc_b_fcc_dense en302208_dense
```

8.2 SETUP.ADVANCED

setup.advanced.preferred_frequencies

Sets preferred frequencies to use.

Class	`VAR`	
Type	`list`	
	default value: 0	
Permissions	guest	`r`
	admin	`rw`

This variable holds a list of the preferred frequencies (in kHz) that the reader should use. It is valid only when the the `setup.region` is set to ETSI or JAPAN, or when `setup.sub_region` is `FCC_PART90_DENSE`, `FCC_PART90_LOWBAND`, `KOREA_MODE_3`, `KOREA_MODE_3_DENSE`, `SINGAPORE_BAND_1`, `SINGAPORE_BAND_1_DENSE`, `HONGKONG_BAND_1`, `HONGKONG_BAND_1_DENSE`, `AUSTRALIA_PART90_DENSE`, `BRAZIL_LICENSED`, `TAIWAN_PART90_DENSE`, `THAILAND_PART90_DENSE.`

The following example sets the preferred frequencies to 865900, 866300 and 866700 for reader:

```
>>> setup.advanced.preferred_frequencies =
865900

866300 866700

ok
```

9 INFO NAMESPACE

9 INFO NAMESPACE

9.1 INFO

info.board_number

Reader manufacturing board number.

Class	`VAR`	
Type	`string`	
	default value: `""`	
Permissions	guest	`r`
	admin	`r`

This variable contains the manufacturing board number of the reader.

info.description

Reader description.

Class	`VAR`	
Type	`string`	
	default value: `"unknown"`	
Permissions	guest	`rw`
	admin	`rw`

This variable contains the assigned description of the reader.

The following example sets the reader description to "Neology Toll RFID Reader set for vehicle processing":

```
>>> info.description = Neology Toll RFID
Reader set for

vehicle processing

ok
```

9 INFO NAMESPACE

info.list_time_zones

Lists available time zone strings

Class	`FUNCTION`	
Parameters	**dir** :	`STRING`
Response	`STRING`	
Permissions	guest	x
	admin	x

Lists the selectable time zones that can be used in the `info.time_zone` variable. Output is provided as a directory listing. Those output strings within square brackets "[…]" can be thought of as directories and used as an input into this function for further listings. Those outputs without the square brackets can be used directly in the `info.time_zone` variable using its relative pathname (`e.g.` "US/Eastern", "US/Central", …)).

The following example shows a time zone listing.

```
>>> info.list_time_zones()

ok

CET CST6CDT EET EST EST5EDT GB GMT GMT+0 GMT-
0 GMT0 Greenwich

HST MET MST MST7MDT NZ NZ-CHAT PRC PST8PDT
ROC ROK UCT UTC

Universal W-SU WET Zulu [Africa] [America]
[Asia] [Australia] [Etc]

[Europe] [Pacific]

>>> info.list_time_zones(America)

ok

Anchorage Caracas Chicago Denver Los_Angeles
New_York

Sao_Paulo
```

To set the `info.time_zone` variable with one of these values, specify the string like you would a directory path to the time zone you want to use. For example, to select the "Chicago" time zone underneath the "America" directory, use "America/Chicago" as the name to specify in the `info.time_zone` variable.

```
>>> info.time_zone=America/Chicago

ok
```

9 INFO NAMESPACE

info.location

Reader location.

Class	`VAR`	
Type	`string`	
	default value: `"unknown"`	
Permissions	guest	`rw`
	admin	`rw`

This variable contains the assigned location of the reader.

The following example sets the reader location to "Parking Deck #3":

```
>>> info.location = Parking Deck #3
ok
```

info.make

Reader make.

Class	`VAR`	
Type	`string`	
	default value: `"Unknown"`	
Permissions	guest	`r`
	admin	`r`

This variable contains the make of the reader.

The following example returns the make of the reader:

```
>>> info.make
ok Toll RFID Reader
```

9 INFO NAMESPACE

info.manufacturer

Reader manufacturer name.

Class	`VAR`	
Type	`string`	
	default value: `"Neology"`	
Permissions	guest	`r`
	admin	`r`

This variable contains the name of the reader manufacturer.

The following example returns the reader manufacturer:

```
>>> info.manufacturer
ok Neology
```

info.manufacturer_description

Manufacturer description.

Class	`VAR`	
Type	`string`	
	default value: `"website:www.neology.net"`	
Permissions	guest	`r`
	admin	`r`

This variable contains the manufacturer description (support information).

The following example returns the manufacturer's description:

```
>>> info.manufacturer_description
ok Neology
```

9 INFO NAMESPACE

info.model

Reader model.

Class	`VAR`
Type	`string`
	default value: `"Unknown"`
Permissions	guest `r`
	admin `r`

This variable contains the model of the reader.

The following example returns the reader model:

```
>>> info.model
ok Neology
```

info.name

Reader name.

Class	`VAR`
Type	`string`
	default value: `"unknown"`
Permissions	guest `rw`
	admin `rw`

This variable contains the assigned name of the reader.

The following example sets the reader name to "Reader #32":

```
>>> info.name = Reader #32
ok
```

9 INFO NAMESPACE

info.serial_number

Reader serial number.

Class	`VAR`
Type	`string`
	default value: `"Unknown"`
Permissions	guest `r`
	admin `r`

This variable contains the reader serial number.

The following example returns the reader serial number:

```
>>> info.serial_number
ok 0F17D4AB93062F56
```

info.sub_model

Reader sub model.

Class	`VAR`
Type	`string`
	default value: `"Unknown"`
Permissions	guest `r`
	admin `r`

This variable contains the sub model of the reader.

The following example returns the reader sub model:

```
>>> info.sub_model
ok 1
```

9 INFO NAMESPACE

info.support_contact

Reader support contact informaiton.

Class	VAR	
Type	string	
	default value: ""	
Permissions	guest	r
	admin	rw

This variable contains the support contact of the reader.

info.time

Current reader time.

Class	VAR	
Type	string	
	default value: "2006-01-01T01:01:01"	
Permissions	guest	rw
	admin	rw

This variable contains the current time of the reader.

The following example returns the reader's current local time:

```
>>> info.time
ok 2006-06-21T14:06:21.853
```

info.time_reporting

Reader reporting time zone (local or GMT).

Class	VAR	
Type	enum	
	default value: LOCAL	
Permissions	guest	r
	admin	rw

The variable contains the reader reporting time zone. When set to LOCAL, the reader reports all times in the local time zone as specified in the `info.time_zone` variable. When set to GMT, all times are reported in Greenwich Mean Time (UTC or zulu time).

The following example sets the time reporting to GMT and then to LOCAL:

```
>>> info.time_reporting = GMT
ok
>>> info.time
ok 2006-06-21T18:11:02.853Z
>>> info.time_reporting = LOCAL
ok
>>> info.time
ok 2006-06-21T14:11:09.015
```

info.time_zone

Reader LOCAL time zone (see `info.list_time_zones`)

Class	`VAR`	
Type	`string`	
	default value: `"GMT"`	
Permissions	guest	`r`
	admin	`rw`

The variable contains the actual reader time zone (`e.g.` America/`New_York`, America/Chicago, …). This value is used by the `info.time_reporting` variable as the LOCAL time zone. When `info.time_reporting` is set to LOCAL, the reader reports all times in the time zone specified in this variable. Use `info.list_time_zones` to get a list of valid values to use for this variable.

The following example sets the reporting time to LOCAL and then changes the LOCAL time zone using the `info.time_zone` variable to see the effects that change has on the reported time.

```
>>> info.time_reporting = LOCAL
ok
>>> info.time_zone = America/New_York
ok
>>> info.time
```

```
ok 2006-06-21T14:11:09.015
>>> info.time_zone = America/Chicago
ok
>>> info.time
ok 2006-06-21T13:11:09.015
```

info.unit_number

Reader unit number.

Class	`VAR`	
Type	`string`	
	default value: `""`	
Permissions	guest	`r`
	admin	`r`

This variable contains the unit number of the reader.

info.zone

Reader zone.

Class	`VAR`	
Type	`string`	
	default value: `"unknown"`	
Permissions	guest	`rw`
	admin	`rw`

This variable contains the assigned zone of the reader.

The following example sets the reader zone to "Dock Door #21":

```
>>> info.zone = Dock Door #21
ok
```

9.2 INFO.PS111

info.ps111.tc_agency_data

Agency Data to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0x3fffffff`
Permissions	guest	`r`
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Agency Data field.

info.ps111.tc_agency_id

Agency ID to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0x7f`
Permissions	guest	`r`
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Agency ID field.

9 INFO NAMESPACE

info.ps111.tc_date

Date to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0x1ff`
Permissions	guest	`r`
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Date field, if `modem.protocol.ps111.control.use_dynamic_data` is `false`.

info.ps111.tc_future

Future Data to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0xf`
Permissions	guest	`r`
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Future Data field.

9 INFO NAMESPACE

info.ps111.tc_lane_id

Lane ID(s) to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`list`	
	default value: `0`	
Permissions	guest	`r`
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Lane ID field.

the variable is a list; each element in the list indicates the Lane ID used for the corresponding antenna. The TC Lane ID value is always generated from the list of `lane_id` values defined by `modem.protocol.ps111.control.write_data.tc_lane_id` except when the `use_dynamic_write_data` is `false` AND the user is calling the write() function with explicit user data, in which case the user data value is used.

info.ps111.tc_plaza_id

Plaza ID to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0x7f`
Permissions	guest	`r`
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Plaza ID field.

9 INFO NAMESPACE

info.ps111.tc_seq_num

Seq / Txn Number to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0xffff`
Permissions	guest	`r`
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Seq / Txn Number field, if `modem.protocol.ps111.control.use_dynamic_data` is `false`.

info.ps111.tc_time

Time to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0x1ffff`
Permissions	guest	`r`
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Time field, if `modem.protocol.ps111.control.use_dynamic_data` is `false`.

info.ps111.tc_vehicle_class

Vehicle Class to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0x7ff`
Permissions	guest	`r`
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Vehicle Class field.

info.ps111.tm_date

Date to be written into the Traffic Management ™ tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0x1ff`
Permissions	guest	`r`
	admin	`rw`

If and when Traffic Management data is written to a PS111 tag, the value of this variable will be used to write the Date field if `modem.protocol.ps111.control.use_dynamic_write_data` is `false`.

9 INFO NAMESPACE

info.ps111.tm_reader_id

Reader ID to be written into the Traffic Management ™ tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0xfff`
Permissions	guest	`r`
	admin	`rw`

If and when Traffic Management data is written to a PS111 tag, the value of this variable will be used to write the Reader ID field.

info.ps111.tm_time

Time to be written into the Traffic Management ™ tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0x1ffff`
Permissions	guest	`r`
	admin	`rw`

If and when Traffic Management data is written to a PS111 tag, the value of this variable will be used to write the Time field if `modem.protocol.ps111.control.use_dynamic_write_data` is `false`.

9.3 INFO.T21

info.t21.agency_code

Reader agency code.

Class	`VAR`
Type	`string`
	default value: `"0x0000"`
Permissions	guest `r`
	admin `rw`

This variable contains the agency code data used in certain Title 21 transactions. It should be set to a 16-hex value, eg. `info.agency_code=0xfa34.`

info.t21.amtech_delayed_ack

Amtech delayed ack per T21 lane application

Class	`VAR`
Type	`string`
	default value: `""`
Permissions	guest `r`
	admin `rw`

info.t21.caltrans_read_thresh

Caltrans read threshold per T21 lane application

Class	`VAR`
Type	`string`
	default value: `""`
Permissions	guest `r`
	admin `rw`

9 INFO NAMESPACE

info.t21.discrimination_mode

Discrimination mode per T21 lane application

Class	`VAR`	
Type	`string`	
	default value: `""`	
Permissions	guest	`r`
	admin	`rw`

info.t21.lane_number

Lane number per T21 lane application

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`65535`
Permissions	guest	`r`
	admin	`rw`

info.t21.lanes

Lanes per T21 lane application

Class	`VAR`	
Type	`string`	
	default value: `""`	
Permissions	guest	`r`
	admin	`rw`

9 INFO NAMESPACE

info.t21.operation_mode

Operation mode per T21 lane application

Class	`VAR`	
Type	`string`	
	default value: `""`	
Permissions	guest	`r`
	admin	`rw`

info.t21.poll_sequence

Poll sequence per T21 lane application

Class	`VAR`	
Type	`string`	
	default value: `""`	
Permissions	guest	`r`
	admin	`rw`

info.t21.poll_to_poll_time

Poll to poll time per T21 lane application

Class	`VAR`	
Type	`string`	
	default value: `""`	
Permissions	guest	`r`
	admin	`rw`

9 INFO NAMESPACE

info.t21.reader_id

Reader identification.

Class	`VAR`	
Type	`string`	
	default value: `"0x00000000"`	
Permissions	guest	`r`
	admin	`rw`

This variable contains the reader ID data used in certain Title 21 transactions. It should be set to a 32-bit hex value, eg. `info.reader_id=0xff00b123.`

info.t21.tiris_power_level

Caltrans and tiris power level per T21 lane application

Class	`VAR`	
Type	`string`	
	default value: `""`	
Permissions	guest	`r`
	admin	`rw`

10 VERSION NAMESPACE

10 VERSION NAMESPACE

10.1 VERSION

version.hw

Reader hardware version.

Class	`VAR`	
Type	`string`	
	default value: `""`	
Permissions	guest	`r`
	admin	`r`

This variable contains the reader hardware version.

The following example returns the reader hardware version:

```
>>> version.hw
ok EP
```

version.hw_detail

Reader hardware version details.

Class	`VAR`	
Type	`string`	
	default value: `""`	
Permissions	guest	`-`
	admin	`r`

This variable contains detailed information on the reader’s hardware version.

The following example returns hardware version details:

```
>>> version.hw_detail
ok 0x0002
```

10 VERSION NAMESPACE

version.rollback

Reader rollback software version.

Class	`VAR`	
Type	`string`	
	default value: `""`	
Permissions	guest	`r`
	admin	`r`

This variable contains the version of software that will be installed if a successful `reader.firmware.rollback()` is executed. If rollback software is not available, "None" is returned.

The following example returns the reader software rollback version:

```
>>> version.rollback
ok 1.0.234
```

version.sw

Reader software version.

Class	`VAR`	
Type	`string`	
	default value: `""`	
Permissions	guest	`r`
	admin	`r`

This variable contains the reader software version.

The following example returns the reader software version:

```
>>> version.sw
ok sw = 1.0.68
```

10 VERSION NAMESPACE

version.sw_detail

Reader software version details.

Class	`VAR`
Type	`string`
	default value: `""`
Permissions	guest `–`
	admin `r`

This variable contains detail information on the reader's software version.

The following example returns software version details:

```
>>> version.sw_detail

ok sw = 1.0-68, base_file = 1.0-57, fw = 1.0-
68, dsp = 5.0, fpga

= 0x0015
```

11 COM NAMESPACE

11 COM NAMESPACE

11.1 COM.EVENT

com.event.overflow_backoff_time

Time the reader will stop sending events if event channel full.

Class	`VAR`	
Type	`int`	
	default value: `3`	
Permissions	guest	`r`
	admin	`rw`

This variable contains the time (in seconds) that the reader will back off or stop sending events out an event channel that cannot handle the current events. After the backoff time has expired, the reader will restart sending events.

The following example sets the overflow backoff time to 2 seconds:

```
>>> com.event.overflow_backoff_time = 2
ok
```

11.2 COM.NETWORK

com.network.connection_table

Dump statistics about the connections to the reader.

Class	`FUNCTION`	
Parameters		
Response	`STRING`	
Permissions	guest	`-`
	admin	`x`

This command dumps statistics about past and present the connections to the reader. It displays the total number of cmd/resp connections handled since reader bootup, the total number of event channels handled since reader bootup, the current number of cmd/resp channels connected and the current number of event channels connected. For each current cmd/resp channel it shows the channel id, the login level of the cmd/resp channel, the number of bytes received on the cmd/resp channel, the number of bytes sent on the cmd/resp channel, the connecting IP address, and the connection time in seconds. For each current event channel it shows the channel id, the number of events sent and the total number of bytes sent on the event channel.

The following example shows using the `com.network.connection_table()` command to get the channel statistics from the reader.

```
com.network.connection_table()

ok

Total cmd/resp channels: 2

Total event channels: 5

Current cmd/resp channels: 1

Current event channels: 4

Cmd/Resp Chan: 11 - level 3, bytes rcvd 512,
bytes sent 66,

ip addr 10.1.1.37, connection time 513

Event Chan: 17 - events sent 1, bytes sent 28

Event Chan: 14 - events sent 1, bytes sent 28

Event Chan: 15 - events sent 1, bytes sent 28

Event Chan: 16 - events sent 1, bytes sent 28
```

com.network.dns_servers

DNS servers to be used.

Class	`VAR`	
Type	`string`	
	default value: `""`	
Permissions	guest	`rw`
	admin	`rw`

This variable contains the setting used to set multiple domain name servers on the network for domain name resolution.

The following example sets two domain servers:

```
>>> com.network.dns_servers=10.1.1.1
10.1.1.50

ok
```

11 COM NAMESPACE

com.network.domain_list

List of domain names to search.

Class	`VAR`	
Type	`string`	
	default value: `""`	
Permissions	guest	`rw`
	admin	`rw`

This variable contains multiple domain names/addresses to search.

The following example sets two domain addresses:

```
>>> com.network.domain_list=10.1.1.1
10.1.1.50
ok
```

com.network.domainname

Domain name of the reader.

Class	`VAR`	
Type	`string`	
	default value: `"Unknown"`	
Permissions	guest	`r`
	admin	`rw`

This variable contains the domain name of the reader.

The following example returns the domain name of the reader:

```
>>> com.network.domainname
ok neology.net
```

11 COM NAMESPACE

com.network.hostname

Host name of the reader.

Class	`VAR`	
Type	`string`	
	default value: `"Unknown"`	
Permissions	guest	`r`
	admin	`rw`

This variable contains the host name of the reader.

The following example returns the host name of the reader:

```
>>> com.network.hostname
ok Dock_Door_01
```

com.network.if_status

Returns interface status for network interfaces.

Class	`FUNCTION`	
Parameters		
Response	`STRING`	
Permissions	guest	`-`
	admin	`x`

This diagnostic function displays the network interface status and statistics. Useful for checking TX and RX packet counts, errors, and address info.

```
>>> com.network.if_status()
ok
eth0 Link encap:Ethernet HWaddr
00:23:68:C3:B4:5D
inet addr:169.15.131.104 Bcast:169.15.131.255
Mask:255.255.255.0
inet6 addr: fe80::223:68ff:fec3:b45d/64
Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500
Metric:1
```

```
RX packets:98 errors:0 dropped:0 overruns:0
frame:0
TX packets:13 errors:0 dropped:0 overruns:0
carrier:0
collisions:0 txqueuelen:1000
RX bytes:8171 (7.9 KiB) TX bytes:1827 (1.7
KiB)
Interrupt:8 DMA chan:8
```

com.network.ntp_servers

NTP servers used to synchronize time.

Class	`VAR`	
Type	`string`	
	default value: `""`	
Permissions	guest	`rw`
	admin	`rw`

This variable contains the setting used to synchronize reader time with network time servers. You can enter multiple server addresses separated by a space.

The following example specifies three server addresses:

```
>>> com.network.ntp_servers=10.1.1.1 10.1.1.2
192.10.34.1
ok
```

com.network.ping

Causes reader to ping another machine.

Class	`FUNCTION`	
Parameters	**ip_address**	: `STRING`
Response	`BOOL`	
Permissions	guest	`–`
	admin	`x`

This diagnostic function causes the reader to ping another machine. It is typically used as a diagnostic tool to determine if the reader can connect to another machine. Do not use this function on a running reader or the reader may become unresponsive.

The following example shows a reader can reach a machine at address `10.1.1.244`, but cannot reach a machine at address `10.0.0.21`.

```
>>> com.network.ping(ip_address = 10.1.1.244)
ok address is reachable
>>> com.network.ping(ip_address = 10.0.0.21)
error.network.address_not_reachable
```

com.network.tcpkeepalive

Set if TCP Keep Alive functionality is needed

Class	`VAR`	
Type	`bool`	
	default value: `true`	
Permissions	guest	`r`
	admin	`rw`

If this variable is set to `true`, communications with the reader will be performed with periodic keep alive transmissions to ensure any zombie connections are recognized and cleaned up. By default, this variable is `true`.

The following example turns TCP Keep Alive off:

```
>>> com.network.tcpkeepalive=false
ok
```

com.network.tcpnodelay

Set to `true` to disable Nagle algorithm

Class	`VAR`	
Type	`bool`	
	default value: `false`	
Permissions	guest	`r`
	admin	`rw`

If this variable is set to `true`, the `TCP_NODELAY` socket option will be set on newly created event and cmd/resp channels. This will disable the Nagle algorithm and create lower latency connections at the possible expense of throughput.

The following example turns TCP Nagle algorithm off:

```
>>> com.network.tcpnodelay=true
ok
```

com.network.tcpsynretries

Set if external connection request failures are taking too long.

Class	`VAR`	
Type	`int`	
	default value: `5`	
Permissions	guest	`r`
	admin	`rw`

If applications are attempting to connect to an external host, it can take a long time before that application fails the connection request as it re-attempts connections up to tcpsynretries times. To shorten this time, move this number lower, to increase this time, move it higher. This sets the /proc/sys/net/ipv4/`tcp_syn_retries` variable in linux. The range of values this variable can accept is 0 to 9.

```
>>> com.network.tcpsynretries=1
ok
```

11.3 COM.NETWORK.<N>

com.network.<n>.default_gateway

Default gateway used for first network interface on reader.

Class	`VAR`	
Type	`string`	
	default value: `""`	
Permissions	guest	`r`
	admin	`r`

This variable contains the default gateway used for the first network interface on the reader. Setting the default gateway to `0.0.0.0` will report a default gateway of none.

The following example returns the default gateway of the first network interface on the reader. To set the default gateway, use the `com.network.1.set()` function.

```
>>> com.network.1.default_gateway
ok 10.1.1.1
```

com.network.<n>.ip_address

IP address of first network interface on reader.

Class	VAR	
Type	string	
	default value: ""	
Permissions	guest	r
	admin	r

This variable contains the IP address of the first network interface on the reader.

The following example gets the IP address of the first network interface on the reader. To set the IP address, use the `com.network.1.set()` function.

```
>>> com.network.1.ip_address
ok 10.1.1.243
```

com.network.<n>.ipv6_address

IPV6 addresses of first network interface on reader.

Class	VAR	
Type	string	
	default value: ""	
Permissions	guest	r
	admin	r

This variable contains the IPV6 addresses of the first network interface on the reader.

The following example gets the IPV6 addresses of the first network interface on the reader. To set the IPV6 addresses, use the `com.network.1.set()`

function.

```
>>> com.network.1.ipv6_address
ok fe80::217:9eff:fe00 2001::1
```

com.network.<n>.ipv6_default_gateway

Default IPV6 gateway used for first network interface on reader.

Class	VAR	
Type	string	
	default value: ""	
Permissions	guest	r
	admin	r

This variable contains the default IPV6 gateway used for the first network interface on the reader. Setting the default gateway to "none" will disable the use of the gateway.

The following example returns the default IPV6 gateway of the first network interface on the reader. To set the IPV6 default gateway, use the `com.network.1.set()` function.

```
>>> com.network.1.ipv6_default_gateway
ok 2001::2
```

com.network.<n>.ipv6_method

Method used to acquire IPV6 address.

Class	VAR	
Type	enum	
	default value: radv_only	
Permissions	guest	r
	admin	r

This variable contains the method used to acquire an IPV6 address. This can be static, where the IPV6 address is statically defined for the reader, or `radv_only` where the IPV6 address is acquired from a router advertisement (either router or daemon on server).

The following example returns the method used by the reader to acquire its IPV6 address. To set the method used to acquire an IPV6 address, use the `com.network.1.set()` function.

```
>>> com.network.1.ipv6_method
ok radv_only
```

com.network.<n>.mac_address

MAC Address of the first network interface on the reader.

Class	VAR	
Type	string	
	default value: "00:00:00:00:00:00"	
Permissions	guest	r
	admin	r

This variable contains the MAC Address of the first network interface on the reader.

The following example returns the MAC address of the reader:

```
>>> com.network.1.mac_address
ok 83:F2:01:71:AD:0B
```

com.network.<n>.method

Method used to acquire IP address.

Class	VAR	
Type	enum	
	default value: dhcp	
Permissions	guest	r
	admin	r

This variable contains the method used to acquire an IP address. This can be static, where the IP address is statically defined for the reader, or dhcp where the IP address is acquired from a dhcp server.

The following example returns the method used by the reader to acquire its IP address. To set the method used to acquire an IP address, use the `com.network.1.set()` function.

```
>>> com.network.1.method
```

```
ok dhcp
```

com.network.<n>.set

Specifies the ip acquisition method, ip address, subnet mask and default gateway

Class	FUNCTION		
Parameters	**method**	:	ENUM DEFINITIONS.ENUM.NETWORK.IP_ACQ_METHODS
	ip_address	:	STRING
	subnet_mask	:	STRING
	default_gateway	:	STRING
	ipv6_method	:	ENUM DEFINITIONS.ENUM.NETWORK.IPV6_ACQ_METHODS
	ipv6_address	:	STRING
	ipv6_default_gateway	:	STRING
Response	BOOL		
Permissions	guest	–	
	admin	x	

This function specifies the IP acquisition method, IP address, subnet mask, and default gateway for the first network interface on the reader. The IP acquisition method can be either static or dhcp for IPV4 and static, `radv_only`, or dhcpv6 for IPV6. If static or dhcpv6 are chosen for IPV6, it does not disable router advertisements from being accepted. Router advertisement acceptance are only disabled when IPV6 is disabled.

For IPV4, if the method is static, the IP address, subnet mask, and default gateway must also be specified in the function call. For IPV6, if the method is static, the IPV6 address/netmask and default gateway must be specified.

If the method is static, setting the default gateway to either none or `0.0.0.0` (IPV4 only) will report a default gateway of none.

If the IP acquisition method is dhcp (or `radv_only/dhcpv6` for IPV6), no ip address, subnet mask, or default gateway should be specified in the function call.

if "method" or `ipv6_method` are not specified, IPV4 or IPV6, respectively, will be disabled. However, an IPV4 method or IPV6 (or both) method must be specified or this function will fail. If the user wants no network communications, disconnect the ethernet cable from the reader.

The following example sets the reader to obtain its IP address via dhcp for IPV4 and IPV6 will be disabled. The second example sets a static ipv4 address, subnet mask and default gateway and IPV6 will be disabled.

```
>>> com.network.1.set(method=dhcp)
ok
>>> com.network.1.set(method=static,
ip_address=10.1.1.243,
subnet_mask=255.255.255.0,
default_gateway=10.1.1.1)
ok
```

The following example sets the reader to obtain its IPV6 address via `radv_only` for IPV6 and IPV4 will be disabled. The second example sets a static IPV6 address/netmask and default gateway and IPV4 will be disabled.

```
>>> com.network.1.set(ipv6_method=radv_only)
ok
>>> com.network.1.set(ipv6_method=static,
ipv6_address=2001::2/64,
ipv6_default_gateway=2001::1)
ok
```

The following example sets the reader to obtain its IPV6 address via `radv_only` and the IPV4 address via dhcp. The second example sets a static IPV6 address/netmask and default gateway and a static IPV4 address, netmask and default gateway.

```
>>> com.network.1.set(method=dhcp,
ipv6_method=radv_only)
ok
>>> com.network.1.set(method=static,
ip_address=10.1.1.243,
subnet_mask=255.255.255.0,
default_gateway=10.1.1.1, ipv6_method=static,
ipv6_address=2001::2/64,
ipv6_default_gateway=2001::1)
ok
```

com.network.<n>.settings

Holds all configuration data for the first network interface.

Class	VAR	
Type	string	
	default value: ""	
Permissions	guest	r
	admin	r

This variable holds all network interface 1 configuration data so it can be saved and restored with one variable.

com.network.<n>.subnet_mask

Subnet mask used for first network interface on reader.

Class	VAR	
Type	string	
	default value: ""	
Permissions	guest	r
	admin	r

This variable contains the subnet mask used for the first network interface on the reader.

The following example returns the subnet mask of the first network interface on the reader. To set the subnet mask, use the `com.network.1.set()` function.

```
>>> com.network.1.subnet_mask
ok 255.255.255.0
```

11.4 COM.NETWORK.DISCOVERY

com.network.discovery.autonomous

Enables/disables autonomous discovery

Class	`VAR`	
Type	`bool`	
	default value: `true`	
Permissions	guest	`r`
	admin	`rw`

This variable enables or disables autonomous discovery. If enabled, discovery will send out unsolicited multicast discovery packets every 40 seconds and during initialization and reader mgmt status changes. If disabled, discovery packets will only be sent out when requested by an external source like RST (Reader Startup Tool).

You MUST restart discovery (`reader.service.stop(discovery)`, `reader.service.start(discovery)`) or reboot the reader for this setting to take effect.

com.network.discovery.ipv6_request_address

Multicast address used for discovery requests.

Class	`VAR`	
Type	`string`	
	default value: `"ff04::efc0:0164"`	
Permissions	guest	`r`
	admin	`rw`

This variable contains the multicast address used for discovery requests. The reader will listen to this address for incoming multicast discovery requests.

This setting will not take effect until the discovery process is restarted. This can be done by rebooting the reader or by executing `reader.service.stop(discovery)` followed by `reader.service.start(discovery)`.

You MUST restart discovery (`reader.service.stop(discovery)`, `reader.service.start(discovery)`) or reboot the reader for this setting to take effect.

com.network.discovery.ipv6_response_address

Multicast address used for discovery responses.

Class	`VAR`	
Type	`string`	
	default value: `"ff04::efc0:0165"`	
Permissions	guest	`r`
	admin	`rw`

This variable contains the multicast address used for discovery responses. The reader will send discovery replies to this multicast address.

> This setting will not take effect until the discovery process is restarted. This can be done by rebooting the reader or by executing `reader.service.stop(discovery)` followed by `reader.service.start(discovery)`.

You MUST restart discovery (`reader.service.stop(discovery)`, `reader.service.start(discovery)`) or reboot the reader for this setting to take effect.

com.network.discovery.request_address

Multicast address used for discovery requests.

Class	`VAR`	
Type	`string`	
	default value: `"239.192.1.100"`	
Permissions	guest	`r`
	admin	`rw`

This variable contains the multicast address used for discovery requests. The reader will listen to this address for incoming multicast discovery requests.

This setting will not take effect until the discovery process is restarted. This can be done by rebooting the reader or by executing `reader.service.stop(discovery)` followed by `reader.service.start(discovery)`.

You MUST restart discovery (`reader.service.stop(discovery)`, `reader.service.start(discovery)`) or reboot the reader for this setting to take effect.

com.network.discovery.response_address

Multicast address used for discovery responses.

Class	`VAR`	
Type	`string`	
	default value: `"239.192.1.101"`	
Permissions	guest	`r`
	admin	`rw`

This variable contains the multicast address used for discovery responses. The reader will send discovery replies to this multicast address.

This setting will not take effect until the discovery process is restarted. This can be done by rebooting the reader or by executing `reader.service.stop(discovery)` followed by `reader.service.start(discovery)`.

You MUST restart discovery (`reader.service.stop(discovery)`, `reader.service.start(discovery)`) or reboot the reader for this setting to take effect.

11.5 COM.NETWORK.READER

com.network.reader.cmd_port

Reader command port configuration

Class	`VAR`	
Type	`int`	
	default value:	`50007`
	min	`50010`
	max	`50015`
Permissions	guest	`r`
	admin	`rw`

This command port will be used as the connection point for external applications to talk with reader management. Typically, this port is configured as 50007. This variable allows this to be changed. However, the reader must be rebooted or power cycled in order for the new port number to take effect.

com.network.reader.event_port

Reader event port configuration

Class	`VAR`	
Type	`int`	
	default value:	`50008`
	min	`50010`
	max	`50015`
Permissions	guest	`r`
	admin	`rw`

This event port will be used as the connection point for external applications to obtain reader management events. Typically, this port is configured as 50008. This variable allows this to be changed. However, the reader must be rebooted or power cycled in order for the new port number to take effect.

11.6 COM.NETWORK.READER.PORTS

com.network.reader.ports.enable

Enable reader ports for cmd/resp and event communications (defaults to ports 50007 and 50008).

Class	`VAR`	
Type	`bool`	
	default value: `TRUE`	
Permissions	guest	`r`
	admin	`rw`

11.7 COM.NETWORK.READER.SSL_PORTS

com.network.reader.ssl_ports.enable

Enable reader ssl ports for cmd/resp and event communications (ports 50002 and 50003).

Class	`VAR`	
Type	`bool`	
	default value: `TRUE`	
Permissions	guest	`r`
	admin	`rw`

11.8 COM.NETWORK.SECURITY

com.network.security.https_min_key_strength

SSL encryption minimum supported key length.

Class	VAR	
Type	enum	
	default value: 128	
Permissions	guest	r
	admin	rw

This variable sets the minimum key length used for SSL message encryption.

The following example sets the minimum encryption key length:

```
>>>
com.network.security.https_min_key_strength=2
56
ok
```

com.network.security.https_min_protocol

SSL protocol minimum level support.

Class	VAR	
Type	enum	
	default value: TLSv1_2	
Permissions	guest	r
	admin	rw

This variable sets the minimum protocol level for SSL transactions.

The following example sets the minimum SSL protocol level:

```
>>>
com.network.security.https_min_protocol=TLSv1
_2
ok
```

com.network.security.load_https_crt_key

Specifies the user certificate and key files to be used for HTTPS connections.

Class	`FUNCTION`		
Parameters	**https_crt_file**	:	`STRING`
	https_key_file	:	`STRING`
Response	`BOOL`		
Permissions	guest	`–`	
	admin	`x`	

This function loads customer's certificate and key files for HTTPS operations with the files specified. The system has default certificate and key files. Loading customer files will override the system default files which will override the system default files which are saved for future restoration with C2 command `com.network.security.restore_https_crt_key()`.

there are only one certificate file and one key file. These are the HTTPS server certificate and key files. If customer has a chain of certificates, the single certificate file is made by concatenating all certificates, starting with the server certificate. For example: Server certificate: `server.crt` Chain certificate 1 (this one leads to the server): `chain1.crt` Chain certificate 2 (this one leads to the 1 above): `chain2.crt` The single certificate is made by: cat `chain1.crt` >> `server.crt` cat `chain2.crt` >> `server.crt` The `server.crt` produced is the one to be loaded to reader.

The following example shows a loading of customer certificate and key files for https

```
>>>
com.network.security.load_https_crt_key(https
_crt_file="/tmp/user.crt",
https_key_file="/tmp/user.key")
ok
```

The following example shows a loading of customer certificate only for https

```
>>>
com.network.security.load_https_crt_key(https
_crt_file="/tmp/user.crt")
ok
```

The following example shows a loading of customer key only for https

```
>>>
com.network.security.load_https_crt_key(https
_key_file="/tmp/user.key")
ok
```

com.network.security.restore_https_crt_key

Restores the default HTTPS certificate and key.

Class	FUNCTION	
Parameters		
Response	BOOL	
Permissions	guest	–
	admin	x

Customer can use the C2 command `com.network.security.load_https_crt_key` to load their own HTTPS certifcate and key. If desired, the system default HTTPS certificate and key can be restored by issuing this command. Doing so, customer’s certificate and key will be removed.

The following example shows restoration of system default HTTPS certificate and key

```
>>>
com.network.security.restore_https_crt_key()
ok
```

11.9 COM.NETWORK.SYSLOG.REMOTE.<N>

com.network.syslog.remote.<n>.ip_address

IP address of remote syslog file server.

Class	`VAR`	
Type	`string`	
	default value: `""`	
Permissions	guest	`r`
	admin	`rw`

This variable contains the IP address of the specified remote syslog file server.

The following example sets the IP address of the specified remote syslog file server.

```
>>>
com.network.syslog.remote.1.ip_address=10.1.1
.57
ok
```

com.network.syslog.remote.<n>.log_level

Log level associated with specified remote syslog file server.

Class	`VAR`	
Type	`string`	
	default value: `"err"`	
Permissions	guest	`r`
	admin	`rw`

This variable sets the logging level of syslog messages sent to the specified remote syslog file server. Valid levels are: emerg, alert, crit, err, warning, notice, info, debug

The following example sets the logging level for messages sent to the specified remote syslog file server.

```
>>>
com.network.syslog.remote.1.log_level=warning
ok
```

11.10 COM.SERIAL

com.serial.baudrate

Baud rate for serial communication.

Class	VAR	
Type	enum	
	default value: 115200	
Permissions	guest	r
	admin	r

This variable contains the baud rate for serial communication.

The following example returns the baud rate of the serial port. To set the baud rate, use the `com.serial.set()` function.

```
>>> com.serial.baudrate
ok 115200
```

com.serial.console

Set the program that will run on the serial console.

Class	FUNCTION	
Parameters	**program**	: ENUM DEFINITIONS.ENUM.CONSOLE.PROGRAM
Response	BOOL	
Permissions	guest	–
	admin	x

Set the program that will run on the serial console. Call this function with CLI parameter to run the CLI on the console (default mode). Call this function with the NONE parameter to disable running a program on the serial console (allows script to use serial).

if the console is diabled by this command, an ok response will not be issued on the serial console.

The following example disables running console application on the serial port

```
>>> com.serial.console(program=none)
```

com.serial.databits

Data bits for serial communication.

Class	`VAR`	
Type	`enum`	
	default value: `8`	
Permissions	guest	`r`
	admin	`r`

This variable contains the number of databits used for serial communication.

The following example returns the number of data bits used by the serial port. To set the number of data bits, use the `com.serial.set()` function.

```
>>> com.serial.databits
ok 8
```

com.serial.echo

Echo for serial communication.

Class	`VAR`	
Type	`bool`	
	default value: `true`	
Permissions	guest	`r`
	admin	`r`

This variable contains the echo setting used for serial communication.

The following example returns the echo setting for the serial port. To enable or disable the echo, use the `com.serial.set()` function.

```
>>> com.serial.echo
ok true
```

11 COM NAMESPACE

com.serial.parity

Parity for serial communication.

Class	`VAR`	
Type	`enum`	
	default value: `NONE`	
Permissions	guest	`r`
	admin	`r`

This variable contains the parity for serial communication.

The following example returns the parity setting for the serial port. To set the parity, use the `com.serial.set()` function.

```
>>> com.serial.parity
ok none
```

com.serial.rawmode

Mode for serial communication.

Class	`VAR`	
Type	`bool`	
	default value: `false`	
Permissions	guest	`r`
	admin	`rw`

This variable contains the behavior mode for the serial console. When set to `true`, a machine interface is presented on the serial console. In this mode no prompts are sent out of the serial port. When set to `false`, a human interface is presented. In this mode, the reader prompt is sent out of the serial port at the end of each command.

The following example sets the serial port to raw mode (machine interface):

```
>>> com.serial.rawmode = true
ok
```

11 COM NAMESPACE

com.serial.set

Specifies the reader’s serial port settings.

Class	FUNCTION		
Parameters	**baudrate**	:	ENUM DEFINITIONS.ENUM.SERIAL.BAUDRATES
	databits	:	ENUM DEFINITIONS.ENUM.SERIAL.DATABITS
	stopbits	:	ENUM DEFINITIONS.ENUM.SERIAL.STOPBITS
	parity	:	ENUM DEFINITIONS.ENUM.SERIAL.PARITY
	echo	:	BOOL
	rawmode	:	BOOL
Response	BOOL		
Permissions	guest	–	
	admin	x	

This function specifies the reader’s serial port setting including: baud rate, data bits, stop bits, parity and echo. rawmode is an optional parameter.

The following example sets the reader’s serial port to 57600 baud, 7 data bits, 1 stop bit, even parity, and echo:

```
>>> com.serial.set(baudrate=57600,
databits=7,
stopbits=1, parity=EVEN, echo=TRUE)
ok
```

com.serial.settings

Holds all serial port configuration data.

Class	VAR	
Type	string	
	default value: ""	
Permissions	guest	r
	admin	r

11 COM NAMESPACE

This variable holds all serial port configuration data so it can be saved and restored with one variable.

com.serial.stopbits

Stop bits for serial communication.

Class	`VAR`	
Type	`enum`	
	default value: `1`	
Permissions	guest	`r`
	admin	`r`

This variable contains the number stopbits for serial communication.

The following example returns the number of stop bits used by the serial port. To set the number of stop bits, use the `com.serial.set()` function.

```
>>> com.serial.stopbits
ok 1
```

12 TAG NAMESPACE

12 TAG NAMESPACE

12.1 TAG

tag.erase

Erases tag fields.

Class	`FUNCTION`		
Parameters	**tag_id**	:	`ARRAY`
	pwd	:	`ARRAY`
	antenna	:	`LIST`
Response	`BOOL`		
Permissions	guest	`x`	
	admin	`x`	

This function erases the fields of a tag. All memory locations supported for a particular tag protocol are erased. The parameters supported by this function are:

- `tag_id` (optional) - Indicates the ID of the tag to erase. If this parameter is not specified, the first tag found in the field will be the tag that is erased.
- `pwd` (optional) - Indicates the access password required to operate on a locked tag (Gen 2 only)
- antenna (optional) - Indicates the list of antennas on which the function executes. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name.

The following example erases a tag:

```
>>> tag.erase(tag_id =
0x306800095EFDDF80000987A5)
ok
```

tag.kill

Kill a tag.

Class	FUNCTION		
Parameters	**tag_id**	:	ARRAY
	kill_pwd	:	ARRAY
	antenna	:	LIST
Response	BOOL		
Permissions	guest	x	
	admin	x	

This function kills a tag. The parameters supported by this function are:

- `tag_id` - Indicates the ID of the tag to kill.
- `kill_pwd` - Indicates the kill password required to kill the tag.
- antenna (optional) - Indicates the list of antennas on which the function executes. If an antenna is not specified, all antennas are tried until the function succeeds or no more antennas are available.

The response to this function is a response name.

The following example kills a tag:

```
>>> tag.kill(tag_id =
0x306800095EFDDF80000987A5,
kill_pwd=0x28F5ACD8)
ok
```

tag.lock

Locks any subset of tag fields.

Class	`FUNCTION`		
Parameters	**lock_fields**	:	`STRING`
	lock_type	:	`ENUM DEFINITIONS.ENUM.TAG.LOCK.TYPES`
	tag_id	:	`ARRAY`
	pwd	:	`ARRAY`
	antenna	:	`LIST`
Response	`BOOL`		
Permissions	guest	`x`	
	admin	`x`	

This function locks any subset of fields on a tag. The parameters supported by this function are:

- `lock_fields` - Indicates the set of tag fields to lock.
- `lock_type` - Indicates the type of lock to apply to the tag's fields.
- `tag_id` (optional) - Indicates the ID of the tag to apply the lock. If this parameter is not specified, the first tag found is locked.
- `pwd` (optional) - Indicates the password required to access a locked tag (Gen 2 only).
- antenna (optional) - Indicates the list of antennas on which the function executes. If an antenna is not specified, all antennas are tried until the function succeeds or no more antennas are available.

The response to this function is a response name.

The following example locks the kill password, access password, and user data of the first tag found in the field:

```
>>> tag.lock(lock_fields = kau,
lock_type=secured)
ok
```

tag.lock_access_pwd

Lock tag access password.

Class	FUNCTION		
Parameters	**lock_type**	:	ENUM DEFINITIONS.ENUM.TAG.LOCK.TYPES
	tag_id	:	ARRAY
	pwd	:	ARRAY
	antenna	:	LIST
Response	BOOL		
Permissions	guest	x	
	admin	x	

This function locks a tag's access password. The parameters supported by this function are:

- `lock_type` - Indicates the type of lock to apply to the tag's access password.
- `tag_id` (optional) - Indicates the ID of the tag to apply the lock. If this parameter is not specified, the first tag found in the field is locked.
- `pwd` (optional) - Indicates the password required to access a locked tag (Gen 2 only).
- antenna (optional) - Indicates the list of antennas on which the function executes. If an antenna is not specified, all antennas are tried until the function succeeds or no more antennas are available.

The response to this function is a response name.

The following example locks the access password of a tag `0x306800095EFDDF80000987A5`:

```
>>>
tag.lock_access_pwd(tag_id=0x306800095EFDDF80
000987A5,
lock_type=secured)
ok
```

tag.lock_id

Lock tag id.

Class	FUNCTION		
Parameters	**lock_type**	:	ENUM DEFINITIONS.ENUM.TAG.LOCK.TYPES
	tag_id	:	ARRAY
	pwd	:	ARRAY
	antenna	:	LIST
Response	BOOL		
Permissions	guest	x	
	admin	x	

This function locks a tag's id. The parameters supported by this function are:

- `lock_type` - Indicates the type of lock to apply to the tag's id.
- `tag_id` (optional) - Indicates the ID of the tag to apply the lock. If this parameter is not specified, the first tag found in the field is locked.
- `pwd` (optional) - Indicates the access password required to operate on a locked tag (Gen 2 only).
- antenna (optional) - Indicates the list of antennas on which the function executes. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name.

The following example permanently locks the ID of tag `0x306800095EFDDF80000987A5`:

```
>>>
tag.lock_id(tag_id=0x306800095EFDDF80000987A5
,
lock_type=perma_locked)
ok
```

tag.lock_kill_pwd

Lock tag kill password.

Class	`FUNCTION`		
Parameters	**lock_type**	:	`ENUM DEFINITIONS.ENUM.TAG.LOCK.TYPES`
	tag_id	:	`ARRAY`
	pwd	:	`ARRAY`
	antenna	:	`LIST`
Response	`BOOL`		
Permissions	guest	`x`	
	admin	`x`	

This function locks a tag's kill password. The parameters supported by this function are:

- `lock_type` - Indicates the type of lock to apply to the tag's kill password.
- `tag_id` (optional) - Indicates the ID of the tag to apply the lock. If this parameter is not specified, the first tag found in the field is locked.
- `pwd` (optional) - Indicates the access password required to operate on a locked tag (Gen 2 only).
- antenna (optional) - Indicates the list of antennas on which the function executes. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name.

The following example secures the kill password of a tag with ID of `0x306800095EFDDF80000987A5` and an access password of `0x892D9F0`:

```
>>>
tag.lock_kill_pwd(tag_id=0x306800095EFDDF8000
0987A5, pwd=0x892D9F0,
lock_type=secured)
ok
```

tag.lock_tid

Lock tag TID segment.

Class	`FUNCTION`		
Parameters	**lock_type**	:	`ENUM DEFINITIONS.ENUM.TAG.LOCK.TYPES`
	tag_id	:	`ARRAY`
	pwd	:	`ARRAY`
	antenna	:	`LIST`
Response	`BOOL`		
Permissions	guest	`x`	
	admin	`x`	

This function locks a tag's TID data. The parameters supported by this function are:

- `lock_type` - Indicates the type of lock to apply to the tag's TID data. .
- `tag_id` (optional) - Indicates the ID of the tag to apply the lock. If this parameter is not specified, the first tag found is locked.
- `pwd` (optional) - Indicates the access password required to operate on a locked tag (Gen 2 only):
- antenna (optional) - Indicates the list of antennas on which the function executes. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name.

The following example locks the TID data of the first tag found:

```
>>> tag.lock_tid(lock_type=secured)
ok
```

tag.lock_user_data

Lock tag user data.

Class	FUNCTION		
Parameters	**lock_type**	:	ENUM DEFINITIONS.ENUM.TAG.LOCK.TYPES
	tag_id	:	ARRAY
	pwd	:	ARRAY
	antenna	:	LIST
Response	BOOL		
Permissions	guest	x	
	admin	x	

This function locks a tag's user data. The parameters supported by this function are:

- `lock_type` - Indicates the type of lock to apply to the tag's user data.
- `tag_id` (optional) - Indicates the ID of the tag to apply the lock. If this parameter is not specified, the first tag found in the field is locked.
- `pwd` (optional) - Indicates the access password required to operate on a locked tag (Gen 2 only):
- antenna (optional) - Indicates the list of antennas on which the function executes. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name.

The following example shows the use of this command to permanently unlock the user data of a tag with ID of `0x306800095EFDDF80000987A5`:

```
>>>
tag.lock_user_data(tag_id=0x306800095EFDDF800
00987A5,
lock_type=perma_unsecured)
ok
```

tag.read

Read all tag fields.

Class	`FUNCTION`		
Parameters	**report**	:	`ENUMLIST DEFINITIONS.ENUM.TAG.READ_FIELDS`
	tag_id	:	`ARRAY`
	pwd	:	`ARRAY`
	antenna	:	`LIST`
Response	`COMPOUND`		
Permissions	guest	`x`	
	admin	`x`	

This function is a generic read function. This can be used to read all fields of the tag. The parameters supported by this function are:

- report - List of fields to read from the tag.
- `tag_id` (optional) - Indicates the ID of the tag from which to read data. If this parameter is not specified, the first tag found in the field is read.
- `pwd` (optional) - Indicates the access password required to read a locked tag (Gen 2 only).
- antenna (optional) - Indicates the list of antennas on which the function executes. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name followed by the tag's data. If the requested fields could not be read, an error is returned.

The following example reads the kill password, access password, and user data from tag `0x306800095EFDDF80000987A5`:

```
>>> tag.read
(tag_id=0x306800095EFDDF80000987A5, report
= kill_pwd access_pwd user_data)
ok kill_pwd=0x0D19F572,
access_pwd=0xA72975B4,
user_data=0x00FD9619028026000A87A5
```

tag.read_access_pwd

Read tag's access password.

Class	FUNCTION		
Parameters	**tag_id**	:	ARRAY
	pwd	:	ARRAY
	antenna	:	LIST
Response	STRING		
Permissions	guest	x	
	admin	x	

This function reads a tag's access password. The parameters supported by this function are:

- `tag_id` - Indicates the ID of the tag from which to read data.
- `pwd` (optional) - Indicates the access password required to read a locked tag (Gen 2 only):
- antenna (optional) - Indicates the list of antennas on which the function executes. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name followed by the tag access password. If the access password is not available in a specific tag type, an error is returned.

The following example reads the access password from tag `0x306800095EFDDF80000987A5`:

```
>>>
tag.read_access_pwd(tag_id=0x306800095EFDDF80
000987A5)
ok access_pwd=0xA72975B4
```

tag.read_id

Read ID of first tag.

Class	`FUNCTION`		
Parameters	**antenna**	:	`LIST`
Response	`STRING`		
Permissions	guest	`x`	
	admin	`x`	

This function reads the ID of the first tag in the field. The parameters supported by this function are:

- antenna (optional) - Indicates the list of antennas on which the function executes. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name, followed by the tag id.

The following example reads the tag identifier of the first tag in the reader's field using antennas 1 and 2:

```
>>> tag.read_id(antenna = 1 2)
ok tag_id=0x306800095EFDDF80000987A5
```

tag.read_kill_pwd

Read tag's kill password.

Class	`FUNCTION`		
Parameters	**tag_id**	:	`ARRAY`
	pwd	:	`ARRAY`
	antenna	:	`LIST`
Response	`STRING`		
Permissions	guest	`x`	
	admin	`x`	

This function reads a tag kill password. The parameters supported by this function are:

- `tag_id` - Indicates the id of the tag from which to read the data.
- `pwd` (optional) - Indicates the access password required to read a locked tag

(Gen 2 only).

- antenna (optional) - Indicates the list of antennas on which the function executes. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name followed by the tag's kill password. If the kill password is not available in a specific tag type, an error is returned.

The following example reads the kill password from tag `0x306800095EFDDF80000987A5`:

```
>>>
tag.read_kill_pwd(tag_id=0x306800095EFDDF8000
0987A5)
ok kill_pwd=0x0D19F572
```

tag.read_tid

Read tag's TID.

Class	FUNCTION		
Parameters	**tag_id**	:	ARRAY
	pwd	:	ARRAY
	antenna	:	LIST
Response	STRING		
Permissions	guest	x	
	admin	x	

This function reads a tag TID. The parameters supported by this function are:

- `tag_id` - Indicates the id of the tag from which to read data.
- `pwd` (optional) - Indicates the access password required to read a locked tag (Gen 2 only).
- antenna (optional) - Indicates the list of antennas on which the function executes. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name followed by the tag TID data. If the TID data is not available in a specific tag type, an error is returned.

The following example reads the TID data from tag `0x306800095EFDDF80000987A5`:

```
>>>
tag.read_tid(tag_id=0x306800095EFDDF80000987A
5)
ok tid=0xE2001040
```

tag.read_user_data

Read tag's user data.

Class	`FUNCTION`		
Parameters	**tag_id**	:	`ARRAY`
	pwd	:	`ARRAY`
	antenna	:	`LIST`
Response	`STRING`		
Permissions	guest	`x`	
	admin	`x`	

This function reads a tag's user data. The parameters supported by this function are:

- `tag_id` - Indicates the id of the tag from which to read data.
- `pwd` (optional) - Indicates the access password required to read a locked tag (Gen 2 only).
- antenna (optional) - Indicates the list of antennas on which the function executes. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name followed by the tag user data. If the user data segment is not available in a specific tag type, an error is returned.

The following example shows the use of this command to read the user data from a tag with id of `0x306800095EFDDF80000987A5`:

```
>>>
tag.read_user_data(tag_id=0x306800095EFDDF800
00987A5)
ok user_data=0x00FD9619028026000A87A5
```

tag.unlock

Unlock tag fields.

Class	FUNCTION		
Parameters	**unlock_fields**	:	STRING
	tag_id	:	ARRAY
	pwd	:	ARRAY
	antenna	:	LIST
Response	BOOL		
Permissions	guest	x	
	admin	x	

This function unlocks any subset of tag fields. The parameters supported by this function are:

- `unlock_fields` - Indicates the set of tag fields to unlock.
- `tag_id` (optional) - Indicates the id of the tag to unlock. If this parameter is not specified, the first tag found in the field will be unlocked.
- `pwd` (optional) - Indicates the access password required to operate on a locked tag (Gen 2 only).
- antenna (optional) - Indicates the list of antennas on which the function executes. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available. .

The response to this function is a response name.

The following example unlocks the kill password and access password of the first tag found in the field:

```
>>> tag.unlock(unlock_fields = ka)
ok
```

tag.write

Write and lock any subset of tag fields.

Class	FUNCTION		
Parameters	**new_tag_id**	:	ARRAY
	kill_pwd	:	ARRAY
	access_pwd	:	ARRAY
	tid	:	ARRAY
	user_data	:	ARRAY
	lock_fields	:	STRING
	lock_type	:	ENUM DEFINITIONS.ENUM.TAG.LOCK.TYPES
	tag_id	:	ARRAY
	pwd	:	ARRAY
	antenna	:	LIST
Response	COMPOUND		
Permissions	guest	x	
	admin	x	

This function is a generic write function that writes and optionally locks any subset of fields on a tag. The parameters supported by this function are:

- `new_tag_id` (optional) - Indicates a new id to write to the tag.
- `kill_pwd` (optional) - Indicates the kill password to write to a tag.
- `access_pwd` (optional) - Indicates access password to write a tag.
- tid (optional) - Indicates the TID data to write to a tag.
- `user_data` (optional) - Indicates the user data to write to a tag.
- `lock_fields` (optional) - Indicates which tag fields should be locked after the data is written. K= `kill_pwd` memory, A= `access_pwd` memory, U= `user_data` memory, E= EPC memory, T= TID memory.
- `lock_type` (optional) - Indicates lock type to apply to the `lock_fields.`
- `tag_id` (optional) - Indicates id of the tag to write the data. If this parameter is not specified, the first tag found in the field is written.
- `pwd` (optional) - Indicates the access password required to write a locked tag (Gen 2 only).

- antenna (optional) - Indicates the list of antennas on which the function executes. If an antenna is not specified, all antennas are tried until the function succeeds or no more antennas are available.

This function includes a verify operation and an "ok" response indicates the data written was verified. The response to this function is a response name.

If `tag.reporting.report_write_verify=true`, then the readback verify data is reported as part of the response.

The following example writes a kill password of `0x12345678`, an access password of `0x88776655`, and user data of `0x12345678AABBCCDD` to a tag `0x306800095EFDDF8000000002`. It also permanently locks the kill password and the access password.

```
>>>
tag.write(tag_id=0x306800095EFDDF8000000002,
kill_pwd=0x12345678, access_pwd=0x88776655,
user_data=0x12345678AABBCCDD, lock_fields =
ka,
lock_type=perma_locked)
ok
```

tag.write_access_pwd

Write a tag’s access password.

Class	FUNCTION		
Parameters	**new_access_pwd**	:	ARRAY
	lock_type	:	ENUM DEFINITIONS.ENUM.TAG.LOCK.TYPES
	tag_id	:	ARRAY
	pwd	:	ARRAY
	antenna	:	LIST
Response	COMPOUND		
Permissions	guest	x	
	admin	x	

This function writes a tag access password. The parameters supported by this function are:

- `new_access_pwd` - Indicates the access password to be written to a tag.
- `lock_type` (optional) - Indicates lock type to apply to access password.
- `tag_id` (optional) - Indicates the id of the tag to write the access password. If this parameter is not specified, the first tag found is written.
- `pwd` (optional) - Indicates the access password required to write a locked tag (Gen 2 only).
- antenna (optional) - Indicates the list of antennas on which the function executes. If an antenna is not specified, all antennas are tried until the function succeeds or no more antennas are available.

This function includes a verify operation and an "ok" response indicates the data written was verified. The response to this function is a response name.

If `tag.reporting.report_write_verify=true`, then the readback verify data is reported as part of the response.

The following example writes an access password of `0x1F8DEA90` to the first tag found on antennas 1 and 3:

```
>>> tag.write_access_pwd(new_access_pwd =
0x1F8DEA90,

antenna = 1 3)

ok
```

tag.write_id

Write tag's id.

Class	FUNCTION		
Parameters	**new_tag_id**	:	ARRAY
	lock_type	:	ENUM DEFINITIONS.ENUM.TAG.LOCK.TYPES
	tag_id	:	ARRAY
	pwd	:	ARRAY
	antenna	:	LIST
Response	COMPOUND		
Permissions	guest	x	
	admin	x	

This function writes a tag id. The parameters supported by this function are:

- `new_tag_id` - Indicates the new tag id to write.
- `lock_type` (optional) - Indicates the type of lock to apply to the tag id.
- `tag_id` (optional) - Indicates the id of the tag to write the new tag id. If this parameter is not specified, the first tag found is written.
- `pwd` (optional) - Indicates the access password required to write a locked tag (Gen 2 only).
- antenna (optional) - Indicates the list of antennas on which the function executes. If an antenna is not specified, all antennas are tried until the function succeeds or no more antennas are available.

This function includes a verify operation, so an "ok" response indicates the tag id has been verified. The response to this function is a response name.

If `tag.reporting.report_write_verify=true`, then the readback verify data is reported as part of the response.

The following example writes a new `tag_id` of `0x306800095EFDDF80002` to the first tag found in the field.

```
>>> tag.write_id(new_tag_id =
0x306800095EFDDF80002)
ok
```

tag.write_kill_pwd

Write tag kill password.

Class	FUNCTION		
Parameters	**kill_pwd**	:	ARRAY
	lock_type	:	ENUM DEFINITIONS.ENUM.TAG.LOCK.TYPES
	tag_id	:	ARRAY
	pwd	:	ARRAY
	antenna	:	LIST
Response	COMPOUND		
Permissions	guest	x	
	admin	x	

This function writes a tag kill password. The parameters supported by this function are:

- `kill_pwd` - Indicates the kill password to be written to a tag.
- `lock_type` (optional) - Indicates lock type to apply to the kill password.
- `tag_id` (optional) - Indicates the id of the tag to write the kill password. If this parameter is not specified, the first tag found is written.
- `pwd` (optional) - Indicates the access password required to write a locked tag (Gen 2 only).
- antenna (optional) - Indicates the list of antennas on which the function executes. If an antenna is not specified, all antennas are tried until the function succeeds or no more antennas are available.

This function includes a verify operation and an "ok" response indicates the data written was verified. The response to this function is a response name.

If `tag.reporting.report_write_verify=true`, then the readback verify data is reported as part of the response.

The following example writes kill password `0x42871904` to tag `0x306800095EFDDF8000000002`, using the access `pwd` `0x75297C1F` to lock the kill password:

```
>>> tag.write_kill_pwd(tag_id =

0x306800095EFDDF8000000002, kill_pwd =
0x42871904, access_pwd =
    0x75297C1F, lock_type = secured)
    ok
```

tag.write_tid

Write tag TID.

Class	FUNCTION		
Parameters	**tid**	:	ARRAY
	lock_type	:	ENUM DEFINITIONS.ENUM.TAG.LOCK.TYPES
	tag_id	:	ARRAY
	pwd	:	ARRAY
	antenna	:	LIST
Response	COMPOUND		
Permissions	guest	x	
	admin	x	

This function writes tag TID data. The parameters supported by this function are:

- tid - Indicates the TID data to be written to a tag.
- `lock_type` (optional) - Indicates the type of lock to apply to the TID data.
- `tag_id` (optional) - Indicates the id of the tag to write the TID data. If this parameter is not specified, the first tag found in the field is written.
- `pwd` (optional) - Indicates the access password required to write a locked tag (Gen 2 only).
- antenna (optional) - Indicates the list of antennas on which the function executes. If an antenna is not specified, all antennas are tried until the function succeeds or no more antennas are available.

This function includes a verify operation, so an "ok" response indicates the data written has been verified. The response to this function is a response name.

If `tag.reporting.report_write_verify=true`, then the readback verify data is reported as part of the response.

The following example writes TID data `0xE2001040` to tag `0x306800095EFDDF8000000002`:

```
    >>> tag.write_tid(tag_id =
0x306800095EFDDF8000000002,
tid = 0xE2001040)
ok
```

tag.write_user_data

Write tag user data.

Class	FUNCTION		
Parameters	**user_data**	:	ARRAY
	lock_type	:	ENUM DEFINITIONS.ENUM.TAG.LOCK.TYPES
	tag_id	:	ARRAY
	pwd	:	ARRAY
	antenna	:	LIST
Response	COMPOUND		
Permissions	guest	x	
	admin	x	

This function writes tag user data. The parameters supported by this function are:

- `user_data` - Indicates the user data to be written to a tag.
- `lock_type` (optional)- Indicates the user data to be written to a tag.
- `tag_id` (optional) - Indicates the id of the tag to write the user data. If this parameter is not specified, the first tag found in the field is written.
- `pwd` (optional) - Indicates the access password required to write a locked tag (Gen 2 only).
- antenna (optional) - Indicates the list of antennas on which the function executes. If an antenna is not specified, all antennas are tried until the function succeeds or no more antennas are available.

This function includes a verify operation and an "ok" response indicates the data written was verified. The response to this function is a response name.

If `tag.reporting.report_write_verify=true`, then the readback verify data is reported as part of the response.

The following example writes user data `0x1234567789a` to tag `0x306800095EFDDF8000000002`.

```
>>> tag.write_user_data(tag_id =
0x306800095EFDDF8000000002, user_data =
0x1234567789a)
ok
```

12.2 TAG.DB

tag.db.acknowledge_timeout

If no tag acknowledgements have been received within this time period (seconds), unacknowledged tags will be stored in nonvolatile memory

Class	VAR	
Type	int	
	default value:	10
	min	5
	max	1000
Permissions	guest	r
	admin	rw

tag.db.clear_repeat_count

Clear repeat count of tags in database.

Class	FUNCTION	
Parameters	**tag_id** :	STRING
Response	BOOL	
Permissions	guest	–
	admin	x

Clear the repeat count field in all tags or specific tags stored in the database.

tag.db.clear_stored_tags

Clear all the stored tags from the file system

This will clear all the tags stored in the file system because they weren't acknowledged before the acknowledgement timeout (see `tag.db.acknowledge_timeout` and `tag.db.store_tags` variables). After calling this function, a reader restart will result in an empty database.

Class	`FUNCTION`	
Parameters		
Response	`BOOL`	
Permissions	guest	`-`
	admin	`x`

tag.db.create_entry_on_arrival

If set to `TRUE`, the database will create a new entry for all tags on arrival (including tags that are already in the database)

Class	`VAR`	
Type	`bool`	
	default value: `FALSE`	
Permissions	guest	`r`
	admin	`rw`

If set to `TRUE`, the database will allow the same tag to have multiple entries in the database. Multiple entries will occur if a tag goes through the field at different times (`i.e.` the tag departs and enters the field again at a later time). This can be useful for vehicle applications that use the database and need to know how many different times a particular tag has gone through a tolling gate. The tags with the same EPC ID can be differentiated in this condition by the `audit_record` field in the tag reporting fields.

When this is set to `FALSE`, there will only be one entry for a given tag in the database no matter how many times a tag passes through the tolling gate.

tag.db.get

Returns tags in database.

Class	`FUNCTION`		
Parameters	**tag_id**	:	`STRING`
	active	:	`BOOL`
	audit_record	:	`INT`
	acknowledged	:	`BOOL`
	max	:	`INT`
Response	`COMPOUNDLIST`		
Permissions	guest	`x`	
	admin	`x`	

This function returns tags stored in the tag database. If `tag_id` or `audit_record` parameters are supplied, only the tag matching that ID or audit record will be returned. If status is suppled, only tags with that status value will be returned. Otherwise, all tags in the database will be returned. If max is supplied, no more than 'max' tags will be returned. If active is supplied and set to `true`, only tags currently active in the field will be returned.

The following example returns all tags from the tag database in the reader:

```
>>> tag.db.get()
ok
(tag_id=0x306800095EFDDF8000000002)
(tag_id=0x306800095EFDDF80000040A1)
(tag_id=0x306800095EFDDF80003F7421)
(tag_id=0x306800095EFDDF80000987A5)
Various fields can be reported for each tag
and are defined with
the configuration variable
tag.reporting.taglist_fields.
```

tag.db.get_and_purge

Returns all tags and purges the database.

Class	FUNCTION	
Parameters		
Response	COMPOUNDLIST	
Permissions	guest	x
	admin	x

This function returns all the tags stored in the database and then purges the database.

The following example returns and purges all tags from the tag database.

```
>>> tag.db.get_and_purge()
ok
(tag_id=0x306800095EFDDF8000000002)
(tag_id=0x306800095EFDDF80000040A1)
(tag_id=0x306800095EFDDF80003F7421)
(tag_id=0x306800095EFDDF80000987A5)
Various fields can be reported for each tag
and are defined with
the configuration variable
tag.reporting.taglist_fields.
```

tag.db.max_count

Maximum number of tags stored in the tag database.

Class	VAR	
Type	int	
	default value:	524288
	min	10
	max	524288
Permissions	guest	r
	admin	rw

This variable is the maximum number of unique tags stored in the tag database.

this will force a resizing of memory and clear all tags in database.

The following example sets the reader's tag database to hold 2000 unique tags:

```
>>> tag.db.max_count = 2000
ok
```

tag.db.max_user_data

Maximum number of bytes of user data per tag that can be stored in the database.

Class	VAR	
Type	int	
	default value: 64	
Permissions	guest	r
	admin	r

Maximum number of bytes of user data per tag that can be stored in the database. This is a read only field and is dependent upon the value of `tag.db.max_count`. The two values are inversely proportional, up to a limit. When the tag max count is cut in half, this value will double and vice versa up to a max limit of 1024.

if `tag.db.memory_use` is set to MINIMIZE, this value will always return as 64.

The following example reads the tag database max user data bytes:

```
>>> tag.db.max_user_data
ok 64
```

tag.db.memory_use

Memory use implementation for tag database

If set to FIXED, the total memory used for all `user_data` in the tag database remains the same regardless of the setting `tag.max_count.` In the FIXED case, reductions in `tag.max_count` result in increases in `tag.max_user_data` up to the limit of 1024.

If set to MINIMIZE, the amount of memory used for `user_data` in each entry in the tag database will be kept to the 64 byte minimum. Reductions in the `tag.max_count` will result in corresponding reductions in total memory use. The memory saved by reducing `max_count` will be a minimum of 128 times the number of tags subtracted from `max_count.` (the 128 comes from the 64 bytes of `user_data` saved plus 32 bytes used for TID and EPC memory)

this will force a resizing of memory and clear all tags in database.

Class	`VAR`	
Type	`enum`	
	default value: `minimize`	
Permissions	guest	`r`
	admin	`rw`

tag.db.next_audit_record

Read only variable that returns the audit record number that will be used for the next tag arrival entry.

Class	`VAR`	
Type	`int`	
	default value: `0`	
Permissions	guest	`r`
	admin	`r`

Read only variable that returns the audit record number that will be used for the next tag arrival entry. This number will be in the range from 0-65535. It will rollover to 0 after receiving a new tag when this value is at 65535.

12 TAG NAMESPACE

tag.db.purge

Purge tag database.

Class	`FUNCTION`	
Parameters		
Response	`BOOL`	
Permissions	guest	`x`
	admin	`x`

This function purges the tag database.

The following example purges all tags from the reader's tag database.

```
>>> tag.db.purge()
ok
```

tag.db.scan_tags

Returns all tags in the field.

Class	`FUNCTION`		
Parameters	**ms**	:	`INT`
	block	:	`BOOL`
	antenna	:	`STRING`
Response	`COMPOUNDLIST`		
Permissions	guest	`x`	
	admin	`x`	

When used in blocking mode (the default), this function inventories all tags in the field for the specified number of milliseconds and returns all tags that were found in the field.

When used in non-blocking mode, this function inventories all tags in the field (sending `event.tag.report` events for each tag as it is singulated) for the specified number of milliseconds. At the end of the time interval specified inventoring stops.

When this function is complete it generates the `event.tag.scan_tags_complete` event.

The following example scans for tags in the field for 6 seconds (6000 milliseconds) while blocking (`i.e.` waiting for the 6 seconds to complete):

```
>>> tag.db.scan_tags(6000, true)
ok
(tag_id=0x306800095EFDDF8000000002)
(tag_id=0x306800095EFDDF80000040A1)
(tag_id=0x306800095EFDDF80003F7421)
(tag_id=0x306800095EFDDF80000987A5)
```

The following example scans for tags in the field for 6 seconds (6000 milliseconds) without blocking. It sees the `event.tag.scan_tags_complete` that is sent at the end of the command and then uses the `tag.db.get()` command to get the list of tags that were inventoried during the 6 seconds.

```
>>> tag.db.scan_tags(6000, false)
ok
```

After 6 seconds the `event.tag.scan_tags_complete` event is seen on an event channel.

```
>>> tag.db.get()
ok
(tag_id=0x306800095EFDDF8000000002)
(tag_id=0x306800095EFDDF80000040A1)
(tag_id=0x306800095EFDDF80003F7421)
(tag_id=0x306800095EFDDF80000987A5)
More fields can be reported for each tag. The
fields to be
reported with each tag are defined with the
configuration variable
tag.reporting.taglist_fields.
```

tag.db.set_acknowledged

Set the acknowledged flag status for a specific tag

Class	FUNCTION		
Parameters	**acknowledged**	:	BOOL
	audit_record	:	INT
Response	BOOL		
Permissions	guest	–	
	admin	x	

This function sets the status flag to the specified value for a specific tag. The audit record number must be used to reference the tag.

tag.db.show

Show tag database info.

Class	FUNCTION		
Parameters	**flag**	:	BOOL
Response	COMPOUND		
Permissions	guest	–	
	admin	x	

This function displays the tag database statistics, and, optionally, hashtab distribution.

```
>>> tag.db.show(1)

ok Unique tags : 3 Active tags: 0
Acknowledged tags: 0

unAcknowledged tags: 3 Free tags: 49994 Hash
index 0: 0 Hash index

1: 0

...
```

tag.db.store_tags

If set to `TRUE`, unacknowledged tags will be stored in nonvolatile memory

If a tag remains unacknowledged for longer than `tag.db.acknowledge_timeout` seconds, then it will be stored in nonvolatile memory. Up to 524,288 unacknowledged tags can be stored.

Class	`VAR`	
Type	`bool`	
	default value: `FALSE`	
Permissions	guest	`r`
	admin	`rw`

12.3 TAG.FILTER.<N>

tag.filter.<n>.enable

Enable this filter.

Class	`VAR`	
Type	`bool`	
	default value: `FALSE`	
Permissions	guest	`r`
	admin	`rw`

This variable enables this filter.

The following example enables tag filter 1.

```
>>> tag.filter.1.enable = true
ok
```

tag.filter.<n>.inclusive

Either include tags that match or don't match

Class	`VAR`	
Type	`bool`	
	default value: `TRUE`	
Permissions	guest	`r`
	admin	`rw`

This variable indicates to either include tags that match (Inclusive) or include tags that do not match (Exclusive) the tag filter.

The following example sets tag filter 1 to include tags that match the tag filter.

```
>>> tag.filter.1.inclusive = true
ok
```

tag.filter.<n>.mask

Mask for tag filter.

Class	`VAR`	
Type	`array`	
	default value: `0x00`	
Permissions	guest	`r`
	admin	`rw`

This variable is the mask (as an array of hex bytes) for the tag filter.

The following example sets the mask for tag filter 1 to `0x30FF`.

```
>>> tag.filter.1.mask = 0x30FF
ok
```

tag.filter.<n>.name

Tag filter name.

Class	`VAR`	
Type	`string`	
	default value: `""`	
Permissions	guest	`r`
	admin	`rw`

This variable is the name given to the tag filter.

The following example sets the name for tag filter 1.

```
>>> tag.filter.1.name = pallet filter
ok
```

tag.filter.<n>.pattern

Pattern for tag filter.

Class	`VAR`	
Type	`array`	
	default value: `0x00`	
Permissions	guest	`r`
	admin	`rw`

This variable is the pattern (as an array of hex bytes) for the tag filter.

The following example sets the pattern for tag filter 1 to `0x3017`.

```
>>> tag.filter.1.pattern = 0x3017
ok
```

tag.filter.<n>.protocol

Protocol selector for tag filter.

Class	`VAR`	
Type	`enum`	
	default value: `ALL`	
Permissions	guest	`r`
	admin	`rw`

This variable is the protocol selector for the filer. A specific protocol can be specified or ALL. The default is ALL. If the protocol selector is configured for a specific protocol, then the 'type' field will be automatically added to `tag.reporting.report_fields.`

The following example sets the protocol selector for tag filter 1 to ISOC.

```
>>> tag.filter.1.protocol = ISOC
ok
```

12.4 TAG.FILTER.ANTENNA_CROSS

tag.filter.antenna_cross.enable

Enable/disable antenna crossing events

Class	`VAR`	
Type	`bool`	
	default value: `FALSE`	
Permissions	guest	`r`
	admin	`rw`

This variable enables/disables antenna crossing events. When enabled, antenna crossing events will be generated.

tag.filter.antenna_cross.max_speed

Max speed of tags in deci-meters per second

This value will put restrictions on tag sample rates. Tags not meeting certain sample rates will be ignored.

Class	VAR	
Type	int	
	default value:	10
	min	1
	max	100
Permissions	guest	r
	admin	rw

tag.filter.antenna_cross.performance_metric

Performance metric to set latency of event versus noise handling

Class	VAR	
Type	int	
	default value:	10
	min	1
	max	20
Permissions	guest	r
	admin	rw

This value will allow tradeoffs to be used for different RF environments. A value of 1 here will indicate an RF environment with little noise and no stray/ambient tags in the field. The antenna cross event will have low latency, but may trigger early or trigger on stray tags in a noisy environment with a lot of reflectivity. A value of 20 here will indicate a noisy RF environment with many stray tags. The latency of the antenna crossing event will be at its highest, but the stray tags will be filtered and there should be no `false` crossing events. A value of 10 is the middle of the ground and should be used unless either lower latency or better noise handling is desired. This is the default.

12.5 TAG.REPORTING

tag.reporting.6bitascii_delimiter

The 6 bit ascii delimiter used when tag reports are reported in this manner

If set to "", no delimiter will be used. If set, only one character is allowed.

Class	`VAR`
Type	`string`
	default value: `""`
Permissions	guest –
	admin `rw`

tag.reporting.antenna_cross_fields

Tag fields reported in `event.tag.antenna_cross` events.

Class	`VAR`
Type	`enumlist`
	default value: `tag_id antenna`
Permissions	guest `r`
	admin `rw`

This variable contains the fields to be reported with `event.tag.antenna_cross` events. Supported fields are `tag_id`, type, time, antenna, frequency, and rssi. WARNING: Enabling the tid field in any of the other `tag.reporting.\*_fields` will reduce the performance of the algorithms that generate the `antenna_cross` event.

The following example includes the `tag_id`, time, and antenna with all `event.tag.antenna_cross` events:

```
>>> tag.reporting.antenna_cross_fields =
tag_id time

antenna

ok
```

After setting the `tag.reporting.antenna_cross_fields` to `tag_id` time antenna", the `event.tag.antenna_cross` events will appear as follows:

```
event.tag.antenna_cross
tag_id=0x306800095EFDDF8000000002,

time=2006-06-22T09:54:34.050, antenna=1

event.tag.antenna_cross
tag_id=0x306800095EFDDF80000040A1,

time=2006-06-22T09:54:34.063, antenna=2
```

```
event.tag.antenna_cross
tag_id=0x306800095EFDDF80003F7421,

time=2006-06-22T09:54:34.077, antenna=3

event.tag.antenna_cross
tag_id=0x306800095EFDDF80000987A5,

time=2006-06-22T09:54:34.093, antenna=4
```

tag.reporting.arrive_fields

Tag fields reported in `event.tag.arrive` events.

Class	VAR	
Type	enumlist	
	default value: tag_id antenna time	
Permissions	guest	r
	admin	rw

The reader supports 3 main tag events: `event.tag.report`, `event.tag.arrive`, and `event.tag.depart`. This variable contains the fields to be reported with `event.tag.arrive` events. Supported fields are `tag_id`, tid, `user_data`, `user_data_offset`, type, time, antenna, frequency, rssi, `packet_counter`, `pw_authentic`, `tid_authentic`, `audit_record`, `tag_type_index`, label (antenna label), and `prot_data`.

enabling the tid and/or the `user_data` field will decrease the read rate performance of the reader.

The following example includes the `tag_id`, time, and antenna with all `event.tag.arrive` events:

```
>>> tag.reporting.arrive_fields = tag_id time

antenna

ok
```

After setting the `tag.reporting.arrive_fields` to `tag_id` time antenna", the `event.tag.arrive` events will appear as follows:

```
event.tag.arrive
```

```
tag_id=0x306800095EFDDF8000000002,
first=2006-06-22T09:54:34.050, antenna=1
event.tag.arrive
tag_id=0x306800095EFDDF80000040A1,
first=2006-06-22T09:54:34.063, antenna=2
event.tag.arrive
tag_id=0x306800095EFDDF80003F7421,
first=2006-06-22T09:54:34.077, antenna=3
event.tag.arrive
tag_id=0x306800095EFDDF80000987A5,
first=2006-06-22T09:54:34.093, antenna=4
```

The time field is labeled with the key 'first' because this is the first time the reader inventoried this tag.

tag.reporting.arrive_generation

Defines when the arrive event gets generated.

Class	VAR	
Type	enum	
	default value: no_wait	
Permissions	guest	r
	admin	rw

The arrive event generally occurs when the first tag report is seen. Unfortunately, this often occurs on the edge of the RF field and results in some of the requested fields being blank as they were not able to be determined in time for the first report to be generated.

If this field is set to its default of `NO_WAIT`, the arrival event will occur coincident with the first tag report. Otherwise, the arrival event will be stalled until the requested data is obtained. For example, if `WAIT_FOR_TID` is chosen, the arrival event will not occur until the TID has been obtained via the tag report mechanism. `WAIT_FOR_DATA` stalls the arrival event until the user data has been properly obtained and the `WAIT_FOR_ALL` stalls the event until both the user data and the TID have been obtained properly.

non-ISOC tags don't have TID, so `WAIT_FOR_TID` is only effective for ISOC tags, setting `WAIT_FOR_TID` is ignored for non-ISOC tags.

When this field is changed from `NO_WAIT`, the `event.tag.raw_arrive` may be useful in conjunction with the `event.tag.arrive.` When waiting for an operation to complete, there is no guarantee an arrival event will come out. However, the raw arrival event will always come out coincident with the first tag report.

tag.reporting.depart_fields

Tag fields reported in `event.tag.depart` events.

Class	`VAR`	
Type	`enumlist`	
	default value: `tag_id antenna time`	
Permissions	guest	`r`
	admin	`rw`

The reader supports three main tag events: `event.tag.report`, `event.tag.arrive`, and `event.tag.depart.` This variable contains the fields to be reported with `event.tag.depart` events. Supported fields are `tag_id`, tid, `user_data`, `user_data_offset`, type, time, antenna, repeat, `audit_record`, `min_rssi`, `max_rssi`, `packet_counter`, label (antenna label), and `prot_data.`

enabling the tid and/or the `user_data` field will decrease the read rate performance of the reader.

The following example includes the `tag_id`, time, repeat count, and antenna with all `event.tag.depart` events:

```
>>> tag.reporting.depart_fields = tag_id time
repeat

antenna

ok
```

After setting the `tag.reporting.depart_fields` to `tag_id` time repeat antenna", the `event.tag.depart` events will appear as follows:

```
event.tag.depart
tag_id=0x306800095EFDDF8000000002,
last=2006-06-22T09:58:33.638, antenna=2,
repeat=14
event.tag.depart
tag_id=0x306800095EFDDF80000040A1,
last=2006-06-22T09:58:33.674, antenna=3,
repeat=14
event.tag.depart
tag_id=0x306800095EFDDF80003F7421,
last=2006-06-22T09:58:33.686, antenna=4,
repeat=14
event.tag.depart
tag_id=0x306800095EFDDF80000987A5,
last=2006-06-22T09:58:33.731, antenna=1,
repeat=15
```

The time field is labeled with the key 'last' because this is the last time the reader inventoried this tag. The repeat field indicates the number of times the reader inventoried this tag.

tag.reporting.depart_time

Tag detection delay.

Class	`VAR`	
Type	`int`	
	default value:	`1000`
	min	`100`
	max	`25000`
Permissions	guest	`r`
	admin	`rw`

This variable is the number of milliseconds of not detecting a previously detected tag in the field before generating the `event.tag.depart` event.

The following example sets the time for not detecting a tag to 3 seconds (3000 milliseconds) before declaring the tag has departed the reader field and sending the `event.tag.depart` event.

```
>>> tag.reporting.depart_time = 3000
ok
```

tag.reporting.estimate_fields

Tag fields reported in `event.tag.estimate` events.

Class	`VAR`	
Type	`enumlist`	
	default value: `tag_id`	
Permissions	guest	`r`
	admin	`rw`

Tag estimate fields are used to report a tag position relative to the antennas used to read the tag. Supported fields are `tag_id`, position, and `position_confidence.` Position distance and `position_confidence` are reported in centimeters.

The following example includes the `tag_id`, position, and `position_confidence` with all `event.tag.estimate` events.

```
>>> tag.reporting.estimate_fields = tag_id
position

position_confidence
```

After setting the `tag.reporting.estimate_fields` to `tag_id` position `position_confidence`, the `event.tag.estimate` events will appear as follows:

```
event.tag.estimate
tag_id=0x306800095EFDDF80000040A1,

position=2,3:145 position_confidence=50

event.tag.estimate
tag_id=0x306800095EFDDF8000000002,

position=5,6:430 position_confidence=100

event.tag.estimate
tag_id=0x306800095EFDDF800003F755,
position=1,2:20 position_confidence=50
```

tag.reporting.raw_arrive_fields

Tag fields reported in `event.tag.raw_arrive` event.

Class	`VAR`
Type	`enumlist`
	default value: `tag_id antenna time`
Permissions	guest `r`
	admin `rw`

`event.tag.raw_arrive` can be used in conjunction with `event.tag.arrive` in cases where additional processing is necessary before sending out the `event.tag.arrive` event. For example, when tag security is enabled, the `event.tag.arrive` event will be delayed until tag security operations can be completed as directed. However, the `event.tag.raw_arrive` will immediately be distributed with information on any initial attempts to complete the tag security operations.

The `raw_arrival` event can be used to indicate the tag processing has begun, while waiting for the regular arrival to signal the operations have been completed. If no arrive event is seen, the requested security or wait for data operations could not be completed.

The tag arrive event can also be delayed when `tag.reporting.arrive_generation` is set to something besides `NO_WAIT.`

The following example includes the `tag_id`, time, and antenna with all `event.tag.raw_arrive` events:

```
>>> tag.reporting.raw_arrive_fields = tag_id
time

antenna

ok
```

After setting the `tag.reporting.raw_arrive_fields` to `tag_id` time antenna", the `event.tag.raw_arrive` events will appear as follows:

```
event.tag.raw_arrive
tag_id=0x306800095EFDDF8000000002,
first=2006-06-22T09:54:34.050, antenna=1
event.tag.raw_arrive
tag_id=0x306800095EFDDF80000040A1,
first=2006-06-22T09:54:34.063, antenna=2
event.tag.raw_arrive
tag_id=0x306800095EFDDF80003F7421,
first=2006-06-22T09:54:34.077, antenna=3
event.tag.raw_arrive
tag_id=0x306800095EFDDF80000987A5,
first=2006-06-22T09:54:34.093, antenna=4
```

The time field is labeled with the key 'first' because this is the first time the reader inventoried this tag.

tag.reporting.raw_tag_data

Enable/disable `event.tag.raw` events.

Class	`VAR`	
Type	`bool`	
	default value: `false`	
Permissions	guest	`r`
	admin	`rw`

When set to `true` this variable will cause the reader to send `event.tag.raw` events. This event is generated each time the reader awaits a tag response during inventory, whether successful or not, and contains any data received by the reader, and statistical data associated with the received data.

enabling `event.tag.raw` events will decrease the read rate performance of the reader.

The following example enables `event.tag.raw` events from the reader.

```
>>> tag.reporting.raw_tag_data = true
```

```
    ok
```

After registering to receive `event.tag.raw` events, the `event.tag.raw` events will appear as follows:

```
    event.tag.raw
    raw_data=0x300833B2DDD9014035050007,
    type=ISOC,
    antenna=1, frequency=910750, rssi=-498,
    demod_result=0,
    quality_metric=995
```

tag.reporting.report_field_enable

Enables using `tag.reporting.report_fields` for generating `event.tag.report`

Class	`VAR`	
Type	`bool`	
	default value: `false`	
Permissions	guest	`r`
	admin	`r`

Tag events like `event.tag.arrive` and `event.tag.depart` etc, which use their event report fields (`tag.reporting.arrive_fields`, `tag.reporting.depart_fields` etc) to control what fields are included in the tag events. There is one exception, `event.tag.report`, which will use `tag.reporting.report_fields` to control reported fields only if `tag.reporting.report_field_enable` is set to `true`.

tag.reporting.report_fields

Tag fields reported in event.tag.report events.

Class	VAR
Type	enumlist
	default value: tag_id antenna time
Permissions	guest r
	admin rw

The reader supports 3 main tag events: event.tag.report, event.tag.arrive, and event.tag.depart. This variable contains the fields to be reported with event.tag.report events. Supported fields are tag_id, tid, user_data, user_data_offset, type, time, antenna, frequency, rssi, phase, tx_power, quality_metric, xpc, and prot_data. The reported fields are also the union of the fields set in tag.reporting.report_fields, tag.reporting.arrive_fields, tag.reporting.depart_fields, tag.reporting.raw_arrive_fields, tag.reporting.antenna_cross_fields, and tag.reporting.taglist_fields.

under tag security, tid and user_data will need to be enabled using this variable in order to be displayed in the event.tag.report. Enabling the tid and/or the user_data field will decrease the read rate performance of the reader.

phase is a licensed features that will need to be purchased to be used. The xpc is specific to ISO-C tags. The prot_data is for protocol-specific data and is only used by select protocols.

The following example includes the tag_id, time, tag type and antenna with all event.tag.report events.

```
>>> tag.reporting.report_fields = tag_id time
type
antenna
ok
```

After setting the tag.reporting.report_fields to tag_id time type antenna", the `event.tag.report` events will appear as follows:

```
event.tag.report
tag_id=0x306800095EFDDF8000000002, type=ISOC,
antenna=1, time=2006-06-22T09:47:52.350
event.tag.report
tag_id=0x306800095EFDDF80000040A1, type=ISOC,
antenna=2, time=2006-06-22T09:47:52.373
event.tag.report
tag_id=0x306800095EFDDF80003F7421, type=ISOC,
antenna=3, time=2006-06-22T09:47:52.407
event.tag.report
tag_id=0x306800095EFDDF80000987A5, type=ISOC,
antenna=4, time=2006-06-22T09:47:52.420
```

tag.reporting.report_write_verify

Tag write command reports verify data in response.

Class	`VAR`	
Type	`bool`	
	default value: `false`	
Permissions	guest	`r`
	admin	`rw`

This variable controls whether the read back verification results are reported as part of a "write" command result. If set to `false`, only "ok" or error are reported for the write result. Write commands will typically perform a "read" operation to verify that the data was written correctly to the tag. This variable will control whether the result of the "read" is reported back as part of the "write" command result.

The following example enables reporting of the verification read data after a write command.

```
>>> tag.reporting.report_write_verify = true
ok
>>> tag.write(kill_pwd=12345678,
access_pwd=12345678)
ok
verify_kill_pwd=0x12345678,verify_access_pwd=
0x12345678
```

tag.reporting.taglist_fields

Tag fields reported in tag list responses.

Class	`VAR`	
Type	`enumlist`	
	default value: `tag_id`	
Permissions	guest	`r`
	admin	`rw`

This variable contains the fields reported for each tag in the response to tag list processing commands. Supported fields are `tag_id`, tid, `user_data`, `user_data_offset`, type, time, antenna, repeat, acknowledged, `tid_authentic`, `pw_authentic`, `audit_record`, `min_rssi`, `max_rssi`, `packet_counter`, `tag_type_index`, and `prot_data.`

enabling the tid and/or the `user_data` field will decrease the read rate performance of the reader. The tag list processing commands include: * `tag.db.get()`

- `tag.db.get_and_purge()`
- `tag.db.scan_tags()`

The following example includes the `tag_id`, time, repeat count, and antenna fields for all tags reported in the response to tag list processing commands:

```
>>> tag.reporting.taglist_fields = tag_id
time repeat

antenna

ok
```

After setting the `tag.reporting.taglist_fields` to `tag_id` time repeat antenna, a call to `tag.db.get()` will return the tag list formatted as follows:

```
>>> tag.db.get()

ok
```

```
(tag_id=0x306800095EFDDF8000000002,
first=2006-06-22T10:07:21.592, last=2006-06-
22T10:07:32.096,
antenna=1, repeat=97)
(tag_id=0x306800095EFDDF80000040A1,
first=2006-06-22T10:07:21.653, last=2006-06-
22T10:07:32.041,
antenna=4, repeat=96)
(tag_id=0x306800095EFDDF80003F7421,
first=2006-06-22T10:07:21.640, last=2006-06-
22T10:07:32.039,
antenna=3, repeat=96)
(tag_id=0x306800095EFDDF80000987A5,
first=2006-06-22T10:07:21.604, last=2006-06-
22T10:07:32.002,
antenna=2, repeat=96)
```

The field labeled 'first' indicates the time at which the reader first inventoried the tag. The field labeled 'last' indicates the time at which the reader last inventoried the tag. The field labeled 'repeat' indicates the number of times the reader inventoried the tag.

12.6 TAG.SECURITY

tag.security.authentication_handle_timeout

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`500`
Permissions	guest	`–`
	admin	`rw`

This determines how long the reader will maintain a handle to a *newly* authenticated tag before timing out and restarting the normal inventory rounds. This will be useful for external applications that want to have fast tag security read/write transactions on a newly arrived tag. This value defaults to 0, which means the handle will not be maintained open after authentication.

> this only works on newly arrived and authenticated tags. Tags that have already been authenticated previously will not have their handle maintained. The application may still use the tag security read/write functions as always, but the transaction will not be as fast.

tag.security.enable_priority

Enables higher process priorities when authentication (tid or password) has been enabled.

Class	`VAR`	
Type	`bool`	
	default value: `true`	
Permissions	guest	–
	admin	`rw`

Enables higher process priorities when authentication (tid or password) has been enabled. This may cause user applications to get delayed from receiving events from the reader as the reader takes a higher priority on the process and will consume process cycles if its runnable.

tag.security.log_errors

(Deprecated) Enables logging of errors during authentication process

This should be left `false` except when debugging as errors can typically occur during the authentication process as the tag moves in and out of the field.

> this variable no longer does anything. Please use `tag.security.log_events` in conjunction with `event.tag.security` to get debug information.

Class	`VAR`	
Type	`bool`	
	default value: `false`	
Permissions	guest	–
	admin	`rw`

tag.security.log_events

Enables logging of debug security events

Class	`VAR`	
Type	`bool`	
	default value: `false`	
Permissions	guest	–
	admin	`rw`

Enables logging of events during authentication process. This should be left `false` except when debugging to avoid the overhead. If turned on an `event.tag.security` event will occur whenever tag security operations are attempted on a tag.

tag.security.packet_counter_enable

Class	`VAR`	
Type	`bool`	
	default value: `FALSE`	
Permissions	guest	–
	admin	`rw`

tag.security.password_authentication_enable

Class	VAR	
Type	bool	
	default value: FALSE	
Permissions	guest	–
	admin	rw

tag.security.read_retries

Number of times to try (not retry) read security commands during authentication

Class	VAR	
Type	int	
	default value:	2
	min	1
	max	100
Permissions	guest	–
	admin	rw

Number of times to try (not retry) read security commands during authentication

tag.security.read_user_data

Read tag's user data.

Class	FUNCTION		
Parameters	**tag_id**	:	ARRAY
	antenna	:	INT
	word_ptr	:	INT
	word_count	:	INT
	rel_hdl	:	BOOL
Response	STRING		
Permissions	guest	–	
	admin	x	

This function reads a tag's user data. The parameters supported by this function are:

- `tag_id` (optional) - Indicates the id of the tag from which to read data.
- antenna (optional) - Indicates the number of antenna on which the function executes. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.
- `word_ptr` (optional) - Place where user data will start to be read (and only that amount of `user_data` specified in `word_count` will be read on word boundaries). If this is not specified, this function will behave the same as `tag.read_user_data` except the password will be automatically generated to read from a read protected (SECURE) `user_data` area. Must be specified in conjunction with `word_count.`
- `word_count` (optional) - Number of words to read from `user_data` area starting at `word_ptr.` Must be specified in conjunction with `word_ptr.`
- `rel_hdl` (optional) - Release tag handle. Defaults to `FALSE`. If set to `true`, the handle to the tag, obtained through the use of the `tag.security.authentication_handle_timeout` mechanism, will be released and the modem will begin reading tags again. Set this to `TRUE` when using the `authentication_handle_timeout` on the last command sent to the reader.

The response to this function is a response name followed by the tag user data. If the user data segment is not available in a specific tag type, an error is returned.

This function requires a secure reader license.

> if the key file imported does not allow particular areas of the user data to be read, 0's will be returned.

The following example shows the use of this command to read the user data from a tag with id of `0x306800095EFDDF80000987A5`:

```
>>>
tag.security.read_user_data(tag_id=0x30680009
5EFDDF80000987A5)
ok user_data=0x00FD9619028026000A87A5
```

tag.security.repeat_interval

Causes failed ISOC security to retry.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`INT_MAX`
Permissions	guest	`r`
	admin	`rw`

This value (if non-zero) represents the inverval in seconds at which the reader will attempt to retry ISOC tag authentication, if it has previously failed. A zero value causes the reader to not retry if the initial authentication fails.

> this variable should only be set to a non-zero value in parking, access control, gated tolling, and other applications with stopped or very slow moving tags. When a tag that initially failed authentication is later successfully authenticated, an `event.tag.authentication` event will be generated.

tag.security.retry_cmd_seq_start

Enable/disable retrying of starting cmd sequence.

Class	`VAR`	
Type	`bool`	
	default value: `false`	
Permissions	guest	`r`
	admin	`rw`

When RSSI is below threshold, if tag authentication is not complete, when more tag reports come, reader always retries cmd sequence start after ending it. When RSSI is above threshold, this variable controls whether the retry will happen or not.

each retry of command sequence starting will send the command up to the number of times determined by `tag.security.read_retries` and `tag.security.sequence_retries`.

The following examples enables/disables `tag.security.retry_cmd_seq_start` on the reader.

```
>>> tag.security.retry_cmd_seq_start = true
ok
>>> tag.security.retry_cmd_seq_start = false
ok
```

tag.security.rssi_threshold

RSSI Threshold for authentication process

This value will be used to determine when we have enough signal strength to signal an error condition when authentication fails. Unless otherwise directed by Neology, leave this value to its default.

Class	VAR	
Type	int	
	default value:	-470
	min	-800
	max	-200
Permissions	guest	–
	admin	rw

tag.security.sequence_retries

Number of times to try (not retry) security commands sequence during authentication

Class	VAR	
Type	int	
	default value:	2
	min	1
	max	100
Permissions	guest	–
	admin	rw

Number of times to try (not retry) security commands sequence during authentication

tag.security.tid_authentication_enable

Globally enables/disables TID authentication Requires either a secure reader license or the `tag

security.commision_enable` flag to be true.

Class	VAR	
Type	bool	
	default value: FALSE	
Permissions	guest	–
	admin	rw

tag.security.wait_for_rssi_threshold

Force the reader to wait for the `rssi_threshold` value before performing any authentication functions.

Class	VAR
Type	bool
	default value: FALSE
Permissions	guest –
	admin rw

Force the reader to wait for the `rssi_threshold` value before performing any authentication functions. Unless otherwise directed by Neology, leave this value to its default.

tag.security.write_id

Write tag's id in secure mode (password auto generated internally).

Class	FUNCTION		
Parameters	**new_tag_id**	:	ARRAY
	tag_id	:	ARRAY
	antenna	:	LIST
	rel_hdl	:	BOOL
Response	COMPOUND		
Permissions	guest	–	
	admin	x	

This function writes a tag id in secure mode. The parameters supported by this function are:

- `new_tag_id` - Indicates the new tag id to write.
- `tag_id` (optional) - Indicates the id of the tag to write the new tag id. If this parameter is not specified, the first tag found is written.
- antenna (optional) - Indicates the list of antennas on which the function executes. If an antenna is not specified, all antennas are tried until the function succeeds or no more antennas are available.
- `rel_hdl` (optional) - Release tag handle. Defaults to `FALSE`. If set to `true`, the handle to the tag, obtained through the use of the `tag.security.authentication_handle_timeout`

mechanism, will be released and the modem will begin reading tags again. Set this to `TRUE` when using the `authentication_handle_timeout` on the last command sent to the reader.

This function includes a verify operation, so an "ok" response indicates the tag id has been verified. The response to this function is a response name.

If `tag.reporting.report_write_verify=true`, then the readback verify data is reported as part of the response.

This function requires a secure reader license.

if the key file imported does not allow the ID to be written, an error will be returned.

The following example writes a new `tag_id` of `0x306800095EFDDF80002` to the first tag found in the field.

```
>>> tag.security.write_id(new_tag_id =
0x306800095EFDDF80002)
ok
```

tag.security.write_retries

Number of times to try (not retry) write security commands during authentication

Class	`VAR`		
Type	`int`		
		default value:	`2`
		min	`1`
		max	`100`
Permissions	guest	`–`	
	admin	`rw`	

Number of times to try (not retry) write security commands during authentication

tag.security.write_user_data

Write tag user data in secure mode (password auto generated internally).

Class	FUNCTION		
Parameters	**user_data**	:	ARRAY
	tag_id	:	ARRAY
	antenna	:	INT
	word_ptr	:	INT
	rel_hdl	:	BOOL
Response	COMPOUND		
Permissions	guest	–	
	admin	x	

This function writes tag user data in secure mode. The parameters supported by this function are:

- `user_data` - Indicates the user data to be written to a tag.
- `tag_id` (optional) - Indicates the id of the tag to write the user data. If this parameter is not specified, the first tag found in the field is written.
- antenna (optional) - Indicates the number of antenna on which the function executes. If an antenna is not specified, all antennas are tried until the function succeeds or no more antennas are available.
- `word_ptr` (optional) - Place where user data will start to be written (and only that amount of `user_data` specified will be written on word boundaries). If this is not specified, this function will behave the same as `tag.write_user_data` except the password will be automatically generated to write to a write protected (SECURE) `user_data` area.
- `rel_hdl` (optional) - Release tag handle. Defaults to `FALSE`. If set to `true`, the handle to the tag, obtained through the use of the `tag.security.authentication_handle_timeout` mechanism, will be released and the modem will begin reading tags again. Set this to `TRUE` when using the `authentication_handle_timeout` on the last command sent to the reader.

This function includes a verify operation and an "ok" response indicates the data written was verified. The response to this function is a response name.

If `tag.reporting.report_write_verify=true`, then the readback verify data is reported as part of the response.

This function requires a secure reader license.

> if the key file imported does not allow particular areas of the user data to be written, an error will be returned.

The following example writes user data `0x1234567789a` to tag `0x306800095EFDDF8000000002`.

```
>>> tag.security.write_user_data(tag_id =
0x306800095EFDDF8000000002, user_data =
0x1234567789a)
ok
```

12.7 TAG.SECURITY.KEY_MGMT

tag.security.key_mgmt.import_keys

Import key file

Class	`FUNCTION`		
Parameters	**file**	:	`STRING`
	index	:	`INT`
	key	:	`ARRAY`
	iv	:	`ARRAY`
Response	`BOOL`		
Permissions	guest	–	
	admin	x	

Import the keys to use for tag security. Requires a secure reader license.

tag.security.key_mgmt.remove_keys

Remove all keys and tag types for use in tag security algorithms

Class	`FUNCTION`	
Parameters		
Response	`BOOL`	
Permissions	guest	–
	admin	`x`

12.8 TAG.SECURITY.SECURE_READER

tag.security.secure_reader.enable_security

Enable secure reader functions

Class	`VAR`	
Type	`bool`	
	default value: `FALSE`	
Permissions	guest	–
	admin	`rw`

When setting to `FALSE`, the default setting, allows admin users to access all secure reader functionality as defined by the imported key file. Setting to `TRUE` requires admin users to either login as the secure reader super-user or access the reader from the secure node.

`tag.security.secure_reader.login()` required before setting this value.

tag.security.secure_reader.login

Login to be able to perform secure reader level functions

Class	`FUNCTION`	
Parameters	**pwd** : `STRING`	
Response	`BOOL`	
Permissions	guest	–
	admin	x

tag.security.secure_reader.logout

Logout session for secure reader

Class	`FUNCTION`	
Parameters		
Response	`BOOL`	
Permissions	guest	–
	admin	x

tag.security.secure_reader.reset_secure_node

Resets the secure ip/mac address of a computer node

Class	`FUNCTION`	
Parameters		
Response	`BOOL`	
Permissions	guest	–
	admin	x

Resets / clears the secure ip/mac address to support as a secure node. After calling this function, no nodes will be considered secure and a user will have to login via `secure_reader.login` in order to enable secure reader functions when `tag.security.secure_reader.enable_security` is set to `true`.

after a node is reset, the connection from the node to reader needs to be restarted in order for the reset to be effective.

`tag.security.secure_reader.login()` required before using this function.

tag.security.secure_reader.set_pwd

Change `secure_reader.login` password.

Class	FUNCTION		
Parameters	**pwd**	:	STRING
	new_pwd	:	STRING
Response	BOOL		
Permissions	guest	–	
	admin	x	

Change `secure_reader.login` password

tag.security.secure_reader.set_secure_node

Sets the secure ip/mac address of a computer node

Class	FUNCTION		
Parameters	**ip_address**	:	STRING
	mac_address	:	STRING
Response	BOOL		
Permissions	guest	–	
	admin	x	

Sets the secure ip/mac address. You have to be logged in via the `tag.security.secure_reader.login` in order to use this function. This function will setup a node (a computer/server for example) to be considered secure and allow that node to use all secure reader functions without having to enter in the `tag.security.secure_reader.login` password when `tag.security.secure_reader.enable_security` is set to `true`. Only IPV4 `ip_addresses` are currently supported. Mac addresses need to be specified in a colon separated list.

after a node is set, the connection from the node to reader needs to be restarted in order for the ip/mac address to be effective.

`tag.security.secure_reader.login()` required before using this function.

The following example will set the computer/server at ip address `10.1.1.11` and mac address 01:02:03:04:05:06 to be considered secure. The reader will allow this computer/server to subsequently use secure reader functions without a secure reader password.

```
>>>
tag.security.secure_reader.set_secure_node(ip
_address=10.1.1.11,
mac_address=01:02:03:04:05:06)
ok
```

12.9 TAG.SECURITY.TAG_TYPE

tag.security.tag_type.clear

Clears the `tag_type` fields associated with the index

Therefore, all `tag.security.tag_type.<index>.\*` fields will be cleared.

Class	FUNCTION		
Parameters	**index**	:	INT
Response	BOOL		
Permissions	guest	–	
	admin	x	

tag.security.tag_type.copy

Copies the `tag_type` fields associated with the indices

After completion, the `to_index` fields will be an exact replica of the `from_index`, including keys.

if the `from_index` location is not subsequently cleared and changed, only the lower ordered index will be used on tags that match the `tag_type.`

Class	FUNCTION		
Parameters	**from_index**	:	INT
	to_index	:	INT
Response	BOOL		
Permissions	guest	–	
	admin	x	

12.10 TAG.SECURITY.TAG_TYPE.<N>

tag.security.tag_type.<n>.block_flags

The flags to use for user memory block protection

One byte value for each block.

Class	VAR	
Type	array	
Permissions	guest	–
	admin	r

The flags to use for user memory block protection. A string of hex values. One byte for each block (see num_blocks). Here are the flags:

- 0x01 → block is read locked
- 0x02 → block is write locked
- 0x04 → block not allowed to be read by the lane reader if it is read locked
- 0x08 → block not allowed to be written by the lane reader if it is write locked
- 0x10 → block will be perma unsecured
- 0x20 block will be perma secured

Defaults to 0x00.

when write locking a block, the entire user memory will get write locked for GEN2 tags. The mechanism is here to support future tags that may support individual block write locking.

tag.security.tag_type.<n>.block_size

The number of bytes for each block specified in user area

block_size * num_blocks should equal user memory size

Class	VAR	
Type	int	
	default value:	0
	min	0
	max	128
Permissions	guest	–
	admin	r

tag.security.tag_type.<n>.epc_flags

The flag to use for EPC ID memory block protection

This is a one byte value. It is specified as a hex value with the following meaning for the flags:

- `0x01` → block is read locked
- `0x02` → block is write locked
- `0x04` → block not allowed to be read by the lane reader if it is read locked
- `0x08` → block not allowed to be written by the lane reader if it is write locked
- `0x10` → block will be perma unsecured
- `0x20` block will be perma secured

Defaults to `0x00`.

read locking is not supported for the `epc_flags`, but its there to match the flag settings for the `block_flags.`

Class	`VAR`	
Type	`array`	
	default value: `0x00`	
Permissions	guest	`–`
	admin	`r`

tag.security.tag_type.<n>.key_index_version_offset

Offset, in bytes, from address 0 in user memory bank where the `key_index` and version information reside

Class	`VAR`	
Type	`int`	
	default value: `0`	
Permissions	guest	`–`
	admin	`r`

Offset from address 0 in user memory bank where the `key_index` and version information reside. This area of memory must reside in a block that is not read protected. It takes up one word of memory (2 bytes) and must be word aligned. -1 indicates the `key_index` is always equal 0 and the version is always equal to 1. You can use this value if you don't want a `key_index` and version or if the tag has no user memory to put this information into.

tag.security.tag_type.<n>.label

Returns label for `tag_type` as specified in original imported key file

Class	`VAR`	
Type	`string`	
	default value: `"Unknown"`	
Permissions	guest	`–`
	admin	`r`

tag.security.tag_type.<n>.num_blocks

Specifies the number of blocks to segment user memory into

Class	VAR	
Type	int	
	default value:	0
	min	0
	max	8
Permissions	guest	–
	admin	r

tag.security.tag_type.<n>.packet_counter_enable

Enable packet counter incrementing for this tag

Class	VAR	
Type	bool	
	default value: FALSE	
Permissions	guest	–
	admin	r

Enables packet counter incrementing for this tag

tag.security.tag_type.<n>.packet_counter_offset

Offset, in bytes, from address 0 in user memory bank where the packet counter resides (this is a 16 bit location)

Class	VAR	
Type	int	
	default value: 0	
Permissions	guest	–
	admin	r

Offset, in bytes, from address 0 in user memory bank where the packet counter resides (this is a 16 bit location and must be word aligned).

tag.security.tag_type.<n>.password_authentication_enable

Enable password authentication for this tag.

Class	VAR	
Type	bool	
	default value: FALSE	
Permissions	guest	–
	admin	r

tag.security.tag_type.<n>.tid_mask

Specifies the mask for the TID of the described tag

Class	VAR	
Type	array	
Permissions	guest	–
	admin	r

Specifies the mask to use against all tag TIDs to see if the tag is a member of the specified tag type (see `tid_value`). This is a hex string (without 0x preamble) that must be of length 2`tid_size.`

tag.security.tag_type.<n>.tid_size

Specifies the size of the TID for the described tag in bytes

Class	VAR	
Type	int	
	default value:	0
	min	0
	max	256
Permissions	guest	–
	admin	r

Specifies the size of the TID. A value of 0 effectively disables the `tag_type` from being used in the security application. This size should be equal to the size of the mask and value pairs and must include the serialized portion of the TID if the TID is being used as the unique identifier in the `unique_id` variable. This size *must* include the serialized portion of the TID as that is the portion of the TID that will be used for password generation.

tag.security.tag_type.<n>.tid_value

Specifies the value for the TID of the described tag

Class	`VAR`	
Type	`array`	
Permissions	guest	–
	admin	`r`

Specifies the value to use against all tag TIDs to see if the tag is a member of the specified tag type (see `tid_mask`). This is a hex string (without 0x preamble) that must be of length 2`tid_size.` After masking the TID of a reported tag, if the resulting value matches this value, the tag is considered to be of the tag type mentioned. result = TID `tid_mask;` `br if (result == tid_value)` → use the description of this `tag_type` for the incoming `tag`…

tag.security.tag_type.<n>.unique_id

Value used to generate unique passwords

Default to TID, but UII (EPC ID) can be used for tags that do not have serialized TID's.

Class	`VAR`	
Type	`enum`	
Permissions	guest	–
	admin	`r`

tag.security.tag_type.<n>.version

Version number for tag

Class	VAR	
Type	int	
	default value:	0
	min	0
	max	255
Permissions	guest	–
	admin	r

This number defines the version of the tag configuration for a particular tag TID type. This allows tag populations with the same TID mask/value set to have different sets of security configurations as new tags come into the field and newer tag security features are desired or come available.

12.11 TAG.SECURITY.TAG_TYPE.<N>.FILTER.<N>

tag.security.tag_type.<n>.filter.<n>.enable

Enables use of `tag_type` filter.

Class	VAR	
Type	bool	
	default value: FALSE	
Permissions	guest	–
	admin	r

tag.security.tag_type.<n>.filter.<n>.offset

Offset, in bytes, from address 0 in user memory bank where the filter resides

Class	VAR	
Type	int	
	default value: 0	
Permissions	guest	–
	admin	r

Offset, in bytes, from address 0 in user memory bank where the filter resides (this is a 16 bit location and must be word aligned).

tag.security.tag_type.<n>.filter.<n>.value

Value for filter to match when deciding which `tag_type` to use for tag security.

Class	VAR	
Type	int	
	default value: 0	
Permissions	guest	–
	admin	r

Value for filter to match when deciding which `tag_type` to use for tag security.

12.12 TAG.WRITEBACK.ISOC

tag.writeback.isoc.bytes_per_write

Bytes per write of the writeback write only operation

A 0 value indicates the entire data (`tag.writeback.isoc.write.data`) array will be written in one shot. Other values will indicate how much data will be written per attempt. This will help in poor RF environments where long writes might be an issue.

Class	VAR	
Type	int	
	default value: 0	
Permissions	guest	r
	admin	rw

tag.writeback.isoc.enable

If set to TRUE, the tag writeback for ISOC tags will be enabled

Class	VAR	
Type	bool	
	default value: FALSE	
Permissions	guest	r
	admin	rw

If set to TRUE, tag writeback features will be enabled. (The `tag.writeback.\*` namespace will be enabled)

tag.writeback.isoc.log_events

If set to TRUE, the `event

tag.writeback` status events will come out

Class	VAR	
Type	bool	
	default value: FALSE	
Permissions	guest	r
	admin	rw

tag.writeback.isoc.read_retries

Number of read retries to make for each read command sent to modem

Class	`VAR`	
Type	`int`	
	default value:	`2`
	min	`1`
	max	`10`
Permissions	guest	`r`
	admin	`rw`

tag.writeback.isoc.sequence_retries

Number of sequence retries to make for each tag report that matches writeback filter

Class	`VAR`	
Type	`int`	
	default value:	`2`
	min	`1`
	max	`10`
Permissions	guest	`r`
	admin	`rw`

tag.writeback.isoc.timeout_on_success

The timeout (milliseconds) following a successful write when the same tag will again wait for commands

After a successful write to a tag, other tags may be found and written until the timeout expires. A maximum of four tags at one time can be associated with the timeout. Setting this value to 0 disables the timeout.

Class	VAR	
Type	int	
	default value:	3000
	min	0
	max	60000
Permissions	guest	r
	admin	rw

tag.writeback.isoc.use_block_write

Use block writes for ISOC tags.

Class	VAR	
Type	bool	
	default value: TRUE	
Permissions	guest	r
	admin	rw

tag.writeback.isoc.write_retries

Number of write retries to make for each write command sent to modem

Class	VAR	
Type	int	
	default value:	2
	min	1
	max	10
Permissions	guest	r
	admin	rw

12.13 TAG.WRITEBACK.ISOC.BASIC

tag.writeback.isoc.basic.filter_mask

Specifies the mask for the TID or UII of the tags targeted for writeback.

Class	`VAR`	
Type	`array`	
	default value: `0x0000`	
Permissions	guest	`r`
	admin	`rw`

Specifies the mask to use against all tag TID's or UII's to see if the tag is a member of the specified tag type (see `filter_value`). This is a hex string (without 0x preamble) that must be of length 2`filter_size.`

tag.writeback.isoc.basic.filter_type

Filter type of the ISOC writeback feature.

Class	`VAR`	
Type	`enum`	
	default value: `all`	
Permissions	guest	`r`
	admin	`rw`

If writeback has been enabled, this variable specifies the type of filtering that will be used to determine what tags to trigger the writeback on. The values are TID, UII, or ALL. If ALL is selected, all ISOC tags will be written back to. If ALL is selected, you can use other filtering mechanisms provided to control writeback. If TID or UII are selected, then the `tag.writeback.isoc.basic.filter_\*` variables will also need to be configured.

tag.writeback.isoc.basic.filter_value

Specifies the value for the TID or UII of the tags targeted for writeback.

Class	`VAR`	
Type	`array`	
	default value: `0x0000`	
Permissions	guest	`r`
	admin	`rw`

Specifies the value to use against all tag TIDs to see if the tag is a member of the specified tag type (see `filter_mask`). This is a hex string (without 0x preamble) that must be of length 2`filter_size.` After masking the TID or UII of a reported tag, if the resulting value matches this value, the tag is considered to be of the tag type to be targeted for writeback. result = TID (or UII) `filter_mask;` `br if (result == filter_value)` → target this tag for writeback.

tag.writeback.isoc.basic.rssi_threshold

RSSI Threshold for ISO-C writeback during basic ops.

Class	`VAR`	
Type	`int`	
	default value:	`-470`
	min	`-800`
	max	`-200`
Permissions	guest	–
	admin	`rw`

RSSI Threshold for ISO-C writeback. This value will be used to determine when we have enough signal strength to attempt an ISO-C writeback for basic ops. Unless otherwise directed by Neology, leave this value to its default.

tag.writeback.isoc.basic.wait_for_rssi_threshold

Force the reader to wait for the `rssi_threshold` value before performing any ISO-C writeback basic ops.

Class	`VAR`	
Type	`bool`	
	default value: `FALSE`	
Permissions	guest	–
	admin	`rw`

Force the reader to wait for the `rssi_threshold` value before performing any any ISO-C writeback basic ops. Unless otherwise directed by Neology, leave this value to its default.

tag.writeback.isoc.basic.wait_for_user_data

No write until `user_data` is read or reported.

Class	`VAR`	
Type	`bool`	
	default value: `FALSE`	
Permissions	guest	`r`
	admin	`rw`

Do not write to tag if `user_data` field has not been read. This prevents older data from being overwritten unless the old `user_data` has been successfully read. Enabling this variable will add `user_data` to tag,`reporting.report_fields` variable if it does not already contain it.

12.14 TAG.WRITEBACK.ISOC.BASIC.IBTTA

tag.writeback.isoc.basic.ibtta.agency_id

Agency ID data to be written back.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`4095`
Permissions	guest	`r`
	admin	`rw`

Agency ID data (0 to 4095) to be written back in IBTTA format.

tag.writeback.isoc.basic.ibtta.lane_id

Lane ID(s) data to be written back.

Class	`VAR`	
Type	`list`	
	default value: `0`	
Permissions	guest	`r`
	admin	`rw`

Lane ID data to be written back in IBTTA format. If a single value is specified, it will be used for all antennas. If a list of values is specified, the first element will be used for antenna 1, the second for antenna 2, etc.

only 5 bit values (0-31) for each lane ID are supported.

tag.writeback.isoc.basic.ibtta.new_tag_window

Perform write when tag hasn't been written within this time.

Class	`VAR`	
Type	`int`	
	default value: `60`	
Permissions	guest	`r`
	admin	`rw`

This time, in seconds, defines the new tag window. A write will be performed only when (a) the timestamp in the tag is older than (now - `new_tag_window`) OR (b) the plaza ID in the tag is different than the reader plaza ID. The point is to only write a tag once as it passes through each plaza, even if it is read multiple times by (possibly) multiple readers.

tag.writeback.isoc.basic.ibtta.plaza_id

Plaza ID data to be written back.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`127`
Permissions	guest	`r`
	admin	`rw`

Plaza ID data (0 to 127) to be written back in IBTTA format.

12.15 TAG.WRITEBACK.ISOC.BASIC.OP.<N>

tag.writeback.isoc.basic.op.<n>.action

Tag writeback action for selected index.

Class	`VAR`	
Type	`enum`	
	default value: `WRITE`	
Permissions	guest	`r`
	admin	`rw`

ADD, IBTTA, SUBTRACT, TIME, and WRITE are the current operations available.

ADD: The value in tag.writeback.isoc.basic.op.X.value will be added to the user data on the tag at the offset and length specified in tag.writeback.isoc.basic.op.X.offset and tag.writeback.isoc.basic.op.X.mask. For example, a "mask" of 0xffff will perform a 16 bit add of the existing data at "offset" on the tag plus the "value" and write the new result back into the same location on the tag in "offset". Note, the "mask" cannot be bigger than 32 bits for this operation.

IBTTA: Data written to the user data at offset tag.writeback.isoc.basic.op.X.offset is in IBTTA format (56 bits, with trailing 8 bits of 0 → total 64 bits; mask s not used), but only when the tag is seen on a new plaza or it's been a while since it was last written (where the time is set by tag.writeback.isoc.basic.ibtta.new_tag_window). The following variables control the data to be written: * tag.writeback.isoc.basic.ibtta.plaza_id * tag.writeback.isoc.basic.ibtta.lane_id * tag.writeback.isoc.basic.ibtta.agency_id Timestamp and occupancy bits are written/updated automatically.

SUBTRACT: The value in tag.writeback.isoc.basic.op.X.value will be subtracted from the user data on the tag at the offset and length specified in tag.writeback.isoc.basic.op.X.offset and tag.writeback.isoc.basic.op.X.mask. For example, a "mask" of 0xffffffff will perform a 32 bit subtract of the existing data at "offset" on the tag plus the "value" and write the new result back into the same location on the tag in "offset". Note, the "mask" cannot be bigger than 32 bits for this operation.

TIME: The Unix time (seconds since January 1, 1970) will be written to the "offset" (tag.writeback.isoc.basic.op.X.offset) specified on the tag. The mask will specify, up to 32 bits, which bits of the 32 bit Unix time result will be written to the tag. For example, using all 32 bits, you would specify 0xffffffff as the mask. For the bottom 16 bits, you would specify 0xffff in the "mask". For the top 16 bits, you would specify 0xffff0000.

WRITE: Write the value specified by tag.writeback.isoc.basic.op.X.data at location tag.writeback.isoc.basic.op.X.offset using tag.writeback.isoc.basic.op.X.mask.

tag.writeback.isoc.basic.op.<n>.data

Writeback operation data for selected index

Its used for WRITE action. See `tag.writeback.isoc.basic.op.X.action` for details.

Class	`VAR`	
Type	`array`	
	default value: `0x0000`	
Permissions	guest	`r`
	admin	`rw`

tag.writeback.isoc.basic.op.<n>.enable

Class	`VAR`	
Type	`bool`	
	default value: `FALSE`	
Permissions	guest	`r`
	admin	`rw`

If set to `TRUE`, the writeback operation features for this index are enabled. This must be set `FALSE` before changing any of the action, data, value, offset, or mask parameters.

tag.writeback.isoc.basic.op.<n>.mask

Mask for the writeback operation at the indicated index

Size of the mask must be in two byte increments.

Class	`VAR`	
Type	`array`	
	default value: `0x0000`	
Permissions	guest	`r`
	admin	`rw`

Mask for the writeback operation at the indicated index. Size of the mask must be in two byte increments. However, the mask itself can target one or more bits, up to the size of the mask you specified. For example, a mask of `0xffff` will operate on a 16 bit data size and the operation will affect all 16 bits on the tag. A mask of `0xff0000ff` will operate on a 32 bit data size but only effect the first 8 and last 8 bits. The other bits on the tag will be left unchanged.

tag.writeback.isoc.basic.op.<n>.offset

Offset, in bytes, from address 0 in user memory bank where the writeback operation will occur

Must be in two byte increments.

Class	`VAR`	
Type	`int`	
	default value: `0`	
Permissions	guest	`r`
	admin	`rw`

Offset from address 0 in user memory bank where the writeback operation will occur. The size and mask of the write will be specified by `tag.writeback.isoc.basic.op.X.mask` and the operation is specified by `tag.writeback.isoc.basic.op.X.action.`

to get the best performance, ensure that the offsets in subsequent operations are contiguous with the previous operations. For example, if `op.1.offset` is 10 and the size of the operation is 4 bytes, than attempt to have the `op.2.offset` equal to 14. To continue that idea, if the size of the `op.2` operation is 2 bytes, than make the `op.3.offset` equal to 16. This will greatly speedup the completion of all the operations on a particular tag.

tag.writeback.isoc.basic.op.<n>.value

Writeback operation value for selected index

Its used for ADD and SUBTRACT actions. See `tag.writeback.isoc.basic.op.X.action` for details.

Class	VAR	
Type	int	
	default value: 0	
Permissions	guest	r
	admin	rw

12.16 TAG.WRITEBACK.ISOC.SECURITY.TAG_TYPE.<N>.OP.<N>

tag.writeback.isoc.security.tag_type.<n>.op.<n>.action

Tag writeback action for selected index.

Class	VAR	
Type	enum	
	default value: WRITE	
Permissions	guest	r
	admin	rw

ADD, SUBTRACT, TIME, and WRITE are the current operations available.

ADD: The value in tag.writeback.isoc.security.tag_type.Y.op.X.value will be added to the user data on the tag at the offset and length specified in tag.writeback.isoc.security.tag_type.Y.op.X.offset and tag.writeback.isoc.security.tag_type.Y.op.X.mask. For example, a "mask" of 0xffff will perform a 16 bit add of the existing data at "offset" on the tag plus the "value" and write the new result back into the same location on the tag in "offset". Note, the "mask" cannot be bigger than 32 bits for this operation.

SUBTRACT: The value in tag.writeback.isoc.security.tag_type.Y.op.X.value will be subtracted from the user data on the tag at the offset and length specified in tag.writeback.isoc.security.tag_type.Y.op.X.offset and tag.writeback.isoc.security.tag_type.Y.op.X.mask. For example, a "mask" of 0xffffffff will perform a 32 bit subtract of the existing data at "offset" on the tag plus the "value" and write the new result back into the same location on the tag in "offset". Note, the "mask" cannot be bigger than 32 bits for this operation.

TIME: The Unix time (seconds since January 1, 1970) will be written to the "offset" (tag.writeback.isoc.security.tag_type.Y.op.X.offset) specified on the tag. The mask will specify, up to 32 bits, which bits of the 32 bit Unix time result will be written to the tag. For example, using all 32 bits, you would specify 0xffffffff as the mask. For the bottom 16 bits, you would specify 0xffff in the "mask". For the top 16 bits, you would specify 0xffff0000.

WRITE: Write the value specified by tag.writeback.isoc.security.tag_type.Y.op.X.data at location tag.writeback.isoc.security.tag_type.Y.op.X.offset using tag.writeback.isoc.security.tag_type.Y.op.X.mask.

the offset must be 16 bit aligned for ISOC.

tag.writeback.isoc.security.tag_type.<n>.op.<n>.data

Writeback operation data for selected index

Its used for WRITE action. See `tag.writeback.isoc.security.tag_type.Y.op.X.actio n` for details.

Class	`VAR`	
Type	`array`	
	default value: `0x0000`	
Permissions	guest	`r`
	admin	`rw`

tag.writeback.isoc.security.tag_type.<n>.op.<n>.enable

Class	`VAR`	
Type	`bool`	
	default value: `FALSE`	
Permissions	guest	`r`
	admin	`rw`

If set to `TRUE`, the writeback operation features for this index are enabled. This must be set `FALSE` before changing any of the action, data, value, offset, or mask parameters.

> `tag.security.tid_authentication_enable` or `tag.security.password_authentication_enable` must be enabled as well as tag security configured (see `tag.security` namespace).

tag.writeback.isoc.security.tag_type.<n>.op.<n>.mask

Mask for the writeback operation at the indicated index

Size of the mask must be in two byte increments.

Class	`VAR`	
Type	`array`	
	default value: `0x0000`	
Permissions	guest	`r`
	admin	`rw`

Mask for the writeback operation at the indicated index. Size of the mask must be in two byte increments. However, the mask itself can target one or more bits, up to the size of the mask you specified. For example, a mask of `0xffff` will operate on a 16 bit data size and the operation will affect all 16 bits on the tag. A mask of `0xff0000ff` will operate on a 32 bit data size but only effect the first 8 and last 8 bits. The other bits on the tag will be left unchanged.

tag.writeback.isoc.security.tag_type.<n>.op.<n>.offset

Offset, in bytes, from address 0 in user memory bank where the writeback operation will occur

Must be in two byte increments.

Class	`VAR`	
Type	`int`	
	default value: `0`	
Permissions	guest	`r`
	admin	`rw`

Offset from address 0 in user memory bank where the writeback operation will occur. The size and mask of the write will be specified by `tag.writeback.isoc.security.tag_type.Y.op.X.mask` and the operation is specified by `tag.writeback.isoc.security.tag_type.Y.op.X.actio n.`

> to get the best performance, ensure that the offsets in subsequent operations are contiguous with the previous operations. For example, if `op.1.offset` is 10 and the size of the operation is 4 bytes, than attempt to have the `op.2.offset` equal to 14. To continue that idea, if the size of the `op.2` operation is 2 bytes, than make the `op.3.offset` equal to 16. This will greatly speedup the completion of all the operations on a particular tag.

tag.writeback.isoc.security.tag_type.<n>.op.<n>.value

Writeback operation value for selected index

Its used for ADD and SUBTRACT actions. See `tag.writeback.isoc.security.tag_type.Y.op.X.actio n` for details.

Class	`VAR`	
Type	`int`	
	default value: `0`	
Permissions	guest	`r`
	admin	`rw`

12.17 TAG.WRITEBACK.PS111

tag.writeback.ps111.new_tag_window

Defines how long a tag is considered to have already been read.

Class	`VAR`	
Type	`int`	
	default value: `60`	
Permissions	guest	`–`
	admin	`rw`

If a tag is read that has the same TC Agency ID, TC Plaza TD, and TC Lane ID, AND the TC Date and Time are within a certain window of the current date and time, the tag is considered to have already been read. This variable configures the time window, in seconds. (For example a value of 60 means that a tag is considered to have already been read if its TC Date and Time are less than 60 seconds old, and the Agency ID, Plaza ID, and Lane ID match the values of these parameters in the `write_data` namespace. See the `modem.protocol.ps111.control.report_occasion` and `tag.writeback.ps111.write_occasion` for variables that depend on the meaning of the new tag window value. See the `tag.writeback.ps111.write_data` namespace for the Agency ID and other parameters mentioned above.

tag.writeback.ps111.retries

Sets the number of retries attempted for the PS111 protocol.

Class	`VAR`	
Type	`int`	
	default value: `3`	
Permissions	guest	`–`
	admin	`rw`

Whenever a PS111 operation is attempted, this variable controls how many attempts are made at the read or write. A 0 indicates no retries (just try once), a 1 indicates one retry (for a total of two write attempts), etc. If write verification is disabled, this variable has no effect on writes; writes are only attempted once. See the `tag.writeback.ps111.verify_write` variable to enable write verification.

tag.writeback.ps111.use_dynamic_write_data

Enables or disables using dynamic time/date/etc

in PS111 writes.

Class	`VAR`	
Type	`bool`	
	default value: `TRUE`	
Permissions	guest	–
	admin	`rw`

When `false`, the TM Date, TM Time, TC Date, TC Time, and TC Seq Num values written to the tag are the values defined in the corresponding `tag.writeback.ps111.write_data` variable. When `true`, these fields are written with values generated on-the-fly by the reader. The TC Lane ID value is always generated from the list of `lane_id` values defined by `tag.writeback.ps111.write_data.tc_lane_id` except when the `use_dynamic_write_data` is `false` AND the user is calling the write() function with explicit user data, in which case the user data value is used.

tag.writeback.ps111.verify_write

Enables or disabled PS111 write verification.

Class	`VAR`	
Type	`bool`	
	default value: `TRUE`	
Permissions	guest	–
	admin	`rw`

When `true`, each PS111 write is followed by a read to verify the data was written correctly. When `false`, no write verification is performed.

tag.writeback.ps111.write_occasion

Controls when PS111 writes occur.

Class	`VAR`
Type	`enum`
	default value: `WHEN_NEW`
Permissions	guest –
	admin `rw`

This variable is used to control when PS111 writes occur. When set to NEVER, the reader never automatically performs a write - the `modem.protocol.ps111.write()` function is the only way to cause a write. When set to ALWAYS, the reader performs a write to the tag every time a tag is read. When set to `WHEN_NEW`, the reader performs a write only the first time a new tag is seen. See the `tag.writeback.ps111.write_data` namespace to control what data is written to the tag. See the `tag.writeback.ps111.new_tag_window` to define when a tag is considered to be new.

tag.writeback.ps111.write_type

Controls which portion of PS111 tag memory will be written.

Class	`VAR`
Type	`enum`
	default value: `TRAFFIC_AND_TOLL`
Permissions	guest –
	admin `rw`

This variable indicates whether the reader will write to the Traffic Management ™ section of R/W tag memory, the Toll Collection (TC) section, or both, whenever the tag is written.

the protocol dictates the entire tag must be written each time, but the reader will simply echo back the read data for the areas of memory which are not being actively written to.

12.18 TAG.WRITEBACK.PS111.WRITE_DATA

tag.writeback.ps111.write_data.tc_agency_data

Agency Data to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0x3fffffff`
Permissions	guest	–
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Agency Data field.

tag.writeback.ps111.write_data.tc_agency_id

Agency ID to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0x7f`
Permissions	guest	–
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Agency ID field.

tag.writeback.ps111.write_data.tc_date

Date to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0x1ff`
Permissions	guest	–
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Date field, if `tag.writeback.ps111.use_dynamic_data` is `false`.

tag.writeback.ps111.write_data.tc_future

Future Data to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0xf`
Permissions	guest	–
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Future Data field.

tag.writeback.ps111.write_data.tc_lane_id

Lane ID(s) to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`list`	
	default value: `0`	
Permissions	guest	`–`
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Lane ID field.

> the variable is a list; each element in the list indicates the Lane ID used for the corresponding antenna. The TC Lane ID value is always generated from the list of `lane_id` values defined by `tag.writeback.ps111.write_data.tc_lane_id` except when the `use_dynamic_write_data` is `false` AND the user is calling the write() function with explicit user data, in which case the user data value is used.

tag.writeback.ps111.write_data.tc_plaza_id

Plaza ID to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0x7f`
Permissions	guest	`–`
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Plaza ID field.

tag.writeback.ps111.write_data.tc_seq_num

Seq / Txn Number to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0xffff`
Permissions	guest	–
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Seq / Txn Number field, if `tag.writeback.ps111.use_dynamic_data` is `false`.

tag.writeback.ps111.write_data.tc_time

Time to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0x1ffff`
Permissions	guest	–
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Time field, if `tag.writeback.ps111.use_dynamic_data` is `false`.

tag.writeback.ps111.write_data.tc_vehicle_class

Vehicle Class to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0x7ff`
Permissions	guest	–
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Vehicle Class field.

tag.writeback.ps111.write_data.tm_date

Date to be written into the Traffic Management ™ tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0x1ff`
Permissions	guest	–
	admin	`rw`

If and when Traffic Management data is written to a PS111 tag, the value of this variable will be used to write the Date field if `tag.writeback.ps111.use_dynamic_write_data` is `false`.

tag.writeback.ps111.write_data.tm_reader_id

Reader ID to be written into the Traffic Management ™ tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0xfff`
Permissions	guest	–
	admin	`rw`

If and when Traffic Management data is written to a PS111 tag, the value of this variable will be used to write the Reader ID field.

tag.writeback.ps111.write_data.tm_time

Time to be written into the Traffic Management ™ tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0x1ffff`
Permissions	guest	–
	admin	`rw`

If and when Traffic Management data is written to a PS111 tag, the value of this variable will be used to write the Time field if `tag.writeback.ps111.use_dynamic_write_data` is `false`.

13 DIO NAMESPACE

13 DIO NAMESPACE

13.1 DIO.CONTROL

dio.control.3

Control of digital I/O selection.

Class	`VAR`	
Type	`enum`	
	default value: `modem`	
Permissions	guest	`r`
	admin	`rw`

This variable selects the processor that controls the digital I/O output pins. If "modem" is selected, the `modem.dio.out.*` selection will control the output pins. If "iop" is selected, the `dio.out.\*` selection will control the output pins.

13.2 DIO.DEBOUNCE

dio.debounce.<n>

Debounce time (ms) for digital input pin.

Class	`VAR`	
Type	`int`	
	default value:	`30`
	min	`0`
	max	`60000`
Permissions	guest	`r`
	admin	`rw`

This variable contains the Debounce time (in milliseconds) for the digital input pin.

The following example sets the debounce time for digital input pin 1 to 50 ms.

```
>>> dio.debounce.1 = 40
ok
```

13.3 DIO.IN

dio.in.<n>

Digital IO input pin value.

Class	VAR	
Type	int	
	default value:	0
	min	0
	max	1
Permissions	guest	r
	admin	rw

This variable contains the digital IO input pin value. This variable is writeable, but will not actually change the value of the input. Writing this variable triggers `event.dio.in.\*` if the value is different than the current value.

The following example sets digital input pin 1 low, generating `event.dio.in.1` value=0.

```
>>> dio.in.1 = 0
ok
```

dio.in.all

Digital IO input pin values.

Class	VAR	
Type	int	
	default value:	0
	min	0
	max	15
Permissions	guest	r
	admin	rw

This variable contains all digital IO input pin values. This variable is writeable, but will not actually change the value of the input. Writing this variable triggers `event.dio.in.\*` and `event.dio.all` if the value is different than the current value.

The following example sets digital input pins 1 and 3 to high, pins 2 and 4 to low. This will generate `event.dio.in.n` value=x for any pin that changes value and generate `event.dio.all` in=`0x5` out=`0x0`.

```
>>> dio.in.all = 0x5
ok
```

13.4 DIO.IN.ALARM.LOGIC_LEVEL

dio.in.alarm.logic_level.<n>

The digital input logic level used for alarm timeout comparisons.

Class	VAR	
Type	int	
	default value:	1
	min	0
	max	1
Permissions	guest	–
	admin	rw

The digital input logic level is used along with the corresponding input pin timeout value to determine if an alarm (in the form of an event) should be generated. If a timeout value is set, the input pin is monitored. If the input pin value does not change during the timeout period AND the input pin value matches the alarm logic level, the event `event.dio.in.alarm.timeout.n` (where n is the pin number) is generated. This alarm event generation can be helpful in alerting to the loss of digitial inputs to the reader.

```
>>> dio.in.alarm.logic_level.1 = 1
ok
```

13.5 DIO.IN.ALARM.TIMEOUT

dio.in.alarm.timeout.<n>

The timeout (in seconds) for alarm generation for the digital input pin.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`600`
Permissions	guest	–
	admin	`rw`

The timeout is used along with the corresponding input pin logic level to determine if an alarm (in the form of an event) should be generated. If a timeout value is set, the input pin is monitored. If the input pin value does not change during the timeout period AND the input pin value matches the alarm logic level, the event `event.dio.in.alarm.timeout.n` (where n is the pin number) is generated. This alarm event generation can be helpful in alerting to the loss of digitial inputs to the reader. The maximum timeout value is 600 seconds. A timeout value of 0 diables the alarm event generation.

```
>>> dio.in.alarm.timeout.1 = 30
ok
```

13.6 DIO.OUT

dio.out.<n>

Digital IO output pin value.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`1`
Permissions	guest	`r`
	admin	`rw`

This variable contains the digital IO output pin value. Writing this variable triggers `event.dio.out.\*` if the value is different than the current value.

The following example sets digital outut pin 1 to high, generating `event.dio.out.1` value=1.

```
>>> dio.out.1 = 1
ok
```

dio.out.all

Digital IO output pin values.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`15`
Permissions	guest	`r`
	admin	`rw`

This variable contains all the Digital IO output pin values. Writing this variable triggers `event.dio.out.\*` and `event.dio.all` if the value is different than the current value.

The following example sets digital output pins 1, 2 and 3 to high, pin 4 to low. This generates `event.dio.out.n` value=x for any pin that changes value and generates `event.dio.all` in=`0x5` out=`0x07`.

```
>>> dio.out.all = 0x7
ok
```

14 ANTENNAS NAMESPACE

14 ANTENNAS NAMESPACE

14.1 ANTENNAS

antennas.configuration

Specify mono- configuration of antennas

`ALL_MONOSTATIC` - All antennas configured as monostatic (antenna numbers: 1 2 3 4)

Class	`VAR`	
Type	`enum`	
	default value: `ALL_MONOSTATIC`	
Permissions	guest	`r`
	admin	`rw`

antennas.detected

Antenna ports with antennas connected.

Class	`VAR`	
Type	`list`	
	default value: `0`	
Permissions	guest	`r`
	admin	`r`

This variable contains a list of port numbers. The ports in this list are presumed to have antennas connected, based on the reflected power.

This list is only updated when antenna checks are performed on the port in question. In other words, antenna port X will only be added to or removed from the list when:

1. port X is in the `mux_sequence`,
2. transmission is enabled, and
3. an antenna check on the port is performed.

Because of the limitations described above, most customers will prefer to use the `modem.antennas.perform_check()` function in order to ascertain the most current information about which of the 4 antenna ports have antennas connected.

The following example returns the list of antenna ports that have antennas connected. In this example, antennas are connected to ports 1 and 2, the mux sequence contains 1 and 2, and the reader is in autonomous mode.

```
>>> antennas.detected
ok 1 2
```

antennas.max_computed_conducted_power

Maximum computed conducted power for all antennas

Class	VAR	
Type	int	
	default value:	330
	min	0
	max	330
Permissions	guest	r
	admin	r

Maximum computed transmit power for all antennas. This is the max value that `antennas.\*.computed_conducted_power` can be calculated to be.

The following example sets the maximum computed transmit power for all antennas to 29 dBm.

```
>>> antennas.max_computed_conducted_power =
290
ok
```

14 ANTENNAS NAMESPACE

antennas.max_set_conducted_power

Maximum conducted power for all antennas

Class	VAR	
Type	int	
	default value:	300
	min	0
	max	300
Permissions	guest	r
	admin	r

Maximum transmit power for all antennas. This is the max value that `antennas.\*.conducted_power` can be set to.

The following example sets the maximum transmit power for all antennas to 29 dBm.

```
>>> antennas.max_set_conducted_power = 290
ok
```

antennas.mux_sequence

Specify a list of antenna ports.

Class	VAR	
Type	string	
	default value: "1"	
Permissions	guest	r
	admin	rw

This variable is a list of antennas used every time an operation is performed on all antennas. The list is used to rearrange the order in which antennas are serviced (3 4 1 2). This list also specifies a subset of antenna ports to use (1 3) or to increase the frequency that certain antennas are serviced (1 2 1 3 1 4). Having a single antenna in this list will result in only that antenna being used for most operations.

The following example sets the antenna multiplexing sequence to antennas 1, 3, and 2.

```
>>> antennas.mux_sequence = 1 3 2
ok
```

antennas.port_count

Number of antenna ports on this reader.

Class	`VAR`	
Type	`int`	
	default value: `4`	
Permissions	guest	`r`
	admin	`r`

This variable is the number of antennas that can be attached to the reader.

The following example returns the number of antenna ports on the reader.

```
>>> antennas.port_count
ok 4
```

14.2 ANTENNAS.<N>

antennas.<n>.conducted_power

Transmit power for antenna port.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`330`
Permissions	guest	`r`
	admin	`rw`

This variable is the antenna port transmit power during CW (in tenths of dBm). If this value is 0, the reader automatically calculates the transmit power based on the advanced settings for the antenna.

The following example sets the transmit power for antenna number 1 to 29 dBm.

```
>>> antennas.1.conducted_power = 290
ok
```

14 ANTENNAS NAMESPACE

antennas.<n>.label

Descriptions for antenna port.

Class	VAR	
Type	string	
	default value: "unknown"	
Permissions	guest	r
	admin	rw

This variable contains the descriptions of the antenna port.

antennas.<n>.position

Antenna position list for this port relative to a fixed reference.

Class	VAR	
Type	list	
	default value: 0	
Permissions	guest	r
	admin	rw

Antenna position, in centimeters, relative to a single site reference point, typically a curb, nearest to antenna 1. If the distance is relative to antenna 1, its position value is zero. If more than one antenna is attached to the port using a splitter, then each position is entered for that port. With split antennas, the first is designated with an A suffix and the second with a B suffix.

The following example sets the antenna position for antenna number one to 2 meters.

```
>>> antennas.1.position=200
ok
```

For two antennas on a port using a splitter each position is entered as a list:

```
>>> antennas.2.position=400 583
ok
```

antennas.<n>.rssi_spread

RSSI spread values between the current antenna and the next in the span.

Class	`VAR`	
Type	`list`	
	default value: `200`	
Permissions	guest	`r`
	admin	`rw`

Change in tag RSSI value, in ddB, relative to the current antenna and the next antenna in the span. If there is no next antenna, this value should be zero.

The following example sets the antenna RSSI spread between antenna number one and antenna number two.

```
>>> antennas.1.rssi_spread=200
ok
```

For two antennas on a port using a splitter each RSSI spread value is entered as a list:

```
>>> antennas.2.rssi_spread=200 200
ok
```

14.3 ANTENNAS.<N>.ADVANCED

antennas.<n>.advanced.attenuation

Attenuation level on this port.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`400`
Permissions	guest	`r`
	admin	`rw`

This variable is the attenuation level (in tenths of dB) used to lower the transmit power level on a port.

The following example sets the attenuation level of antenna number 1 to 75 ddB.

```
>>> antennas.1.advanced.attenuation = 75
ok
```

antennas.<n>.advanced.cable_loss

Cable loss on this port.

Class	VAR	
Type	int	
	default value:	18
	min	0
	max	100
Permissions	guest	r
	admin	rw

This variable is the cable loss (in tenths of dB) for the cable(s) connected to this port.

The following example specifies the cable loss for antenna 1 as 15 ddB.

```
>>> antennas.1.advanced.cable_loss = 15
ok
```

antennas.<n>.advanced.computed_conducted_power

Computed conducted power on this port.

Class	VAR	
Type	int	
	default value: 0	
Permissions	guest	r
	admin	r

This variable is the computed conducted power (in tenths of dB) on a port.

The following example shows the advanced method for specifying the antenna data needed to have the reader set the conducted power for antenna number 1. The last command in the example reads the reader computed conducted power.

The computed conducted power is based on the region/`sub_region` and the value of 259 is only presented as an example. Your actual computed conducted power may be different based on the readers region/`sub_region`.

```
>>> antennas.1.conducted_power = 0
ok
>>> antennas.1.advanced.gain_units = dBdC
ok
>>> antennas.1.advanced.gain = 50
ok
>>> antennas.1.advanced.cable_loss = 15
ok
>>> antennas.1.advanced.attenuation = 75
ok
>>>
antennas.1.advanced.computed_conducted_power
ok 259
```

antennas.<n>.advanced.gain

The gain provided by the antenna attached to this port.

Class	VAR	
Type	int	
	default value:	130
	min	-200
	max	200
Permissions	guest	r
	admin	rw

This variable is the gain provided by the antenna attached to this port. The units for the gain are specified by the `antennas.X.advanced.gain_units` variable.

The following example specifies the gain for antenna 1 (a circular antenna) as 50 ddB.

```
>>> antennas.1.advanced.gain_units = dBdC
ok
>>> antennas.1.advanced.gain = 50
ok
```

antennas.<n>.advanced.gain_units

This variable specifies antenna gain units.

Class	VAR	
Type	enum	
	default value: dBi	
Permissions	guest	r
	admin	rw

Antenna Gain.

The following example specifies gain units in dBdC for antenna 1.

```
>>> antennas.1.advanced.gain_units = dBdC
ok
```

14.4 ANTENNAS.CHECK

antennas.check.time

Antenna checks interval.

Class	VAR	
Type	int	
	default value: 1000	
Permissions	guest	r
	admin	rw

This variable is the interval (in ms), for timed antenna checks. If the `antennas.check.type` is set to TIMED, this parameter determines the time interval. If a new antenna port is selected and it is checked in this amount of time, an antenna check is performed.

The following example sets the antenna checking interval to 2000 ms.

```
>>> antennas.check.type = timed
ok
>>> antennas.check.time = 2000
ok
```

antennas.check.type

Determines when antenna checks are performed.

Class	VAR	
Type	enum	
	default value: ALWAYS	
Permissions	guest	r
	admin	rw

This variable determines when the reader checks for an antenna.

- Whenever the antenna port is changed (ALWAYS)
- First time a port is used (FIRST_TIME_ONLY)
- When a port is selected and hasn't been checked for a given period of time (TIMED)

The following example sets the antenna checking to the first time an antenna port is used.

```
>>> antennas.check.type = first_time_only
ok
```

15 MODEM NAMESPACE

15

MODEM NAMESPACE

15.1 MODEM.ANTENNAS

modem.antennas.perform_check

Check each antenna port for connected antenna.

Class	`FUNCTION`	
Parameters		
Response	`STRING`	
Permissions	guest	–
	admin	`x`

This function performs a check on all 4 antenna ports, to determine if an antenna is connected. It returns a list of ports with antennas connected. This function will temporarily disrupt any current RFID transactions.

15.2 MODEM.CONTROL.INVENTORY

modem.control.inventory.perform_rounds

Performs the specified number of inventory rounds.

Class	`FUNCTION`		
Parameters	**rounds**	:	`INT`
	block	:	`BOOL`
	antenna	:	`INT`
Response	`BOOL`		
Permissions	guest	–	
	admin	`x`	

This function is used to perform the number of inventory rounds specified by the "rounds" parameter. If "block" is `true` or unspecified, the function blocks until the rounds are complete before returning. If "block" is `false`, the function returns immediately and does not block. In either case, an `event.status.inventory_rounds_complete` event is generated at the completion of the inventory rounds.

modem.control.inventory.period

The period (microseconds) at which inventory commands are sent.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`INT_MAX`
Permissions	guest	`r`
	admin	`rw`

Inventory commands will be sent every X microseconds, where X is set by this variable. Between inventory cycles will be a period of RF off.

15.3 MODEM.CONTROL.PHASE

modem.control.phase.invert

Invert reported phase values.

Class	`VAR`	
Type	`bool`	
	default value: `FALSE`	
Permissions	guest	`–`
	admin	`rw`

If `true`, this variable will cause the phase reported in tag events to be inverted. (That is, the phase reported will be `0x10000` - [original phase]).

15.4 MODEM.CONTROL.PILOT

modem.control.pilot.max_threshold

Maximum pilot tone detect threshold in dBm

If `modem.control.pilot.threshold` is 0 (automatic), then the computed threshold will be capped at the `max_threshold.` If set to 0, then the max_threshold will not be applied.

Class	VAR	
Type	int	
	default value:	-75
	min	-100
	max	0
Permissions	guest	-
	admin	rw

modem.control.pilot.write_threshold

Pilot tone detect threshold in dBm for writes

Class	VAR	
Type	int	
	default value:	-70
	min	-100
	max	-40
Permissions	guest	-
	admin	rw

15.5 MODEM.CONTROL.SYNC

modem.control.sync.dio_in

Digital input pin to be used to sense the sync signal.

Class	VAR	
Type	int	
	default value:	4
	min	3
	max	4
Permissions	guest	r
	admin	rw

modem.control.sync.dio_out

Digital output pin to be used for the sync signal output.

Class	VAR	
Type	int	
	default value:	4
	min	3
	max	4
Permissions	guest	r
	admin	rw

modem.control.sync.inhibit_antenna_dio3

Transmit antenna to be inhibited by signal on DIO3

If -1 then inhibit disabled on DIO3.

Class	VAR	
Type	int	
	default value:	-1
	min	-1
	max	4
Permissions	guest	r
	admin	rw

modem.control.sync.inhibit_antenna_dio4

Transmit antenna to be inhibited by signal on DIO4

If -1 then inhibit disabled on DIO4.

Class	VAR	
Type	int	
	default value:	-1
	min	-1
	max	4
Permissions	guest	r
	admin	rw

modem.control.sync.mode

Reader synchronization mode

Class	VAR	
Type	enum	
	default value: OFF	
Permissions	guest	r
	admin	rw

Sets the reader synchronization mode. OFF=no synchronization, INHIBIT=reader inhibits using adjacent antenna when signal is active.

modem.control.sync.threshold

Threshold for sync control.

Class	`VAR`	
Type	`int`	
	default value:	`100`
	min	`100`
	max	`300`
Permissions	guest	`r`
	admin	`rw`

15.6 MODEM.DEBUG

modem.debug.channel_scan

Generate `event.status.channel_scan` events.

Class	`VAR`	
Type	`bool`	
	default value: `False`	
Permissions	guest	–
	admin	`rw`

If `True` then `event.status.channel_scan` events will be generated.

modem.debug.channel_scan_coherent

Enable/disable coherent channel scanning.

Class	`VAR`	
Type	`bool`	
	default value: `False`	
Permissions	guest	–
	admin	`rw`

15.7 MODEM.DIAG

modem.diag.current_temperature

Last read temperature in degrees Celsius.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`-20`
	max	`100`
Permissions	guest	`r`
	admin	`r`

This variable stores the last temperature value read from the RF board temperature sensor. This value is writable for test purposes only.

modem.diag.lna_bypass

Bypasses the first LNA stage

When set to -1 it is automatic.

Class	`VAR`	
Type	`int`	
	default value:	`1`
	min	`-1`
	max	`1`
Permissions	guest	`–`
	admin	`rw`

modem.diag.vgadac

Sets the DAC value for the VGA

When set to -1 it is automatic.

Class	`VAR`	
Type	`int`	
	default value:	`512`
	min	`-1`
	max	`1023`
Permissions	guest	–
	admin	`rw`

15.8 MODEM.DIAG.ERROR_HANDLER

modem.diag.error_handler.period

Time period at which the error event filter is flushed.

Class	`VAR`	
Type	`int`	
	default value: `60`	
Permissions	guest	–
	admin	`rw`

The longer this time, the less frequently you get events from the modem when there are multiple error/warning/info events of the same type.

15.9 MODEM.PROTOCOL

modem.protocol.cmd_retries

Number of command retries for failed command.

Class	VAR	
Type	int	
	default value:	2
	min	0
	max	8
Permissions	guest	–
	admin	rw

This variable sets the number of command retries for a failed command. When a command sequence fails, it can be retried for the number of times indicated by this parameter. This is transparent to the user. Each protocol may also have a `cmd_retries` variable for retrying at the air interface command level.

15.10 MODEM.PROTOCOL.ASTMV6.CONTROL

modem.protocol.astmv6.control.extra_inventory_cycles

Number of extra inventory cycles per round for ASTMv6

Class	VAR	
Type	int	
	default value:	0
	min	0
	max	50
Permissions	guest	r
	admin	rw

This specifies the number of extra inventory cycles per round for ASTMv6 protocol. When 0 (default), there is one inventory cycle performed. Each increment of this variable will cause an additional ASTMv6 inventory cycle to be performed during a single round.

15 MODEM NAMESPACE

modem.protocol.astmv6.control.frequency

Frequency to use for ASTMv6 protocol.

Class	`VAR`	
Type	`int`	
	default value:	`915750`
	min	`914250`
	max	`915750`
Permissions	guest	–
	admin	`rw`

The reader will use the specified frequency (in kHz) when performing ASTMv6 operations, then go back to the standard frequency scheme for other protocols. 915750 is the default frequency for this protocol.

modem.protocol.astmv6.control.lane_based_frames

Lane based (instead of wide-area) ASTMv6 frames.

Class	`VAR`	
Type	`bool`	
	default value: `true`	
Permissions	guest	–
	admin	`rw`

When `true`, lane-based ASTMv6 frames are used; when `false` (default), wide-area ASTMv6 frames are used.

modem.protocol.astmv6.control.tx_atten

Transmit attenuation for ASTMv6 in ddBm.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`300`
Permissions	guest	`r`
	admin	`rw`

This specifies the transmit attenuation applied during ASTMv6 transmissions. The value is in ddBm so a value of 200 would be an attenuation of 20 dBm.

15.11 MODEM.PROTOCOL.ASTMV6.FILTER.<N>

modem.protocol.astmv6.filter.<n>.enable

Enable ASTMv6 filter 1.

Class	`VAR`	
Type	`bool`	
	default value: `true`	
Permissions	guest	`-`
	admin	`rw`

When `true`, and if filtering is enabled, only tags which match the specified tag and message type will be reported.

modem.protocol.astmv6.filter.<n>.tag_count

Number of tags filtered (not reported).

Class	`VAR`	
Type	`int`	
	default value: `0`	
Permissions	guest	`r`
	admin	`r`

Count of the number of tag reads not reported due to the filter.

modem.protocol.astmv6.filter.<n>.type

Tag / message type filter 1.

Class	`VAR`	
Type	`int`	
	default value:	`32`
	min	`0`
	max	`255`
Permissions	guest	`-`
	admin	`rw`

Specifies an eight bit tag/transponder and message type. If filtering is enabled, only tags which match this tag type (using bits defined by `type_mask`) will be reported. Example: if type is `0x20` and `type_mask` is `0xFE`, then tags with "Transponder Type" `0x2` (first nibble) and "Message Type" `0x0` or `0x1` (second nibble) will be reported.

modem.protocol.astmv6.filter.<n>.type_mask

Tag / message mask filter 1.

Class	VAR	
Type	int	
	default value:	254
	min	0
	max	255
Permissions	guest	–
	admin	rw

Specifies an eight bit tag/transponder and message mask. If filtering is enabled, this variable defines which bits in the 8 bit tag/message type field will be used to match with the `modem.protocol.astmv6.filter.1.type` value. Example: if type is `0x20` and `type_mask` is `0xFE`, then tags with "Transponder Type" `0x2` (first nibble) and "Message Type" `0x0` or `0x1` (second nibble) will be reported.

15.12 MODEM.PROTOCOL.ASTMV6.FILTERING

modem.protocol.astmv6.filtering.enable

Enable filtering of ASTMv6 IDs based on tag/msg type.

Class	VAR	
Type	bool	
	default value: false	
Permissions	guest	–
	admin	rw

When `true`, `modem.protocol.astmv6.filter.\*` variables are used and only tags with matching tag and message type are reported. This can be useful to prevent some ghost tags in low SNR environments.

15.13 MODEM.PROTOCOL.ASTMV6.PHYSICAL

modem.protocol.astmv6.physical.pilot_threshold

Detection threshold for ASTMv6, in dBm

Class	VAR	
Type	int	
	default value:	-55
	min	-100
	max	0
Permissions	guest	–
	admin	rw

Detection threshold (sensitivity) for ASTMv6 tags. Increase to a higher value to reduce sensitivity, which should also reduce CRC errors ghost tags. A value of 0 indicates the standard/default threshold should be used.

15.14 MODEM.PROTOCOL.CMD_RETRY

modem.protocol.cmd_retry.hop_method

When a tag command fails, frequency hopping control method used

Class	VAR	
Type	enum	
	default value: ALWAYS_HOP	
Permissions	guest	–
	admin	rw

When a tag command fails, and retries are used the reader can use one of three frequency control methods: 1) always hop prior to the retry 2) never hop or 3) use the current noise level to determine if a hop should occur. The `USE_NOISE_LEVEL` method will apply to those sub regions specified by `modem.protocol.cmd_retry.noise_level_sub_regions.` All other sub regions not specified will frequency hop on command failures.

modem.protocol.cmd_retry.noise_thresh

Noise threshold (ddBm) used for command retries.

Class	`VAR`	
Type	`int`	
	default value:	`-650`
	min	`-1200`
	max	`-400`
Permissions	guest	–
	admin	`rw`

Noise threshold used for command retries in ddBm. This is used when `modem.protocol.cmd_retry_hop_method` is `USE_NOISE_LEVEL.` If the current noise level is greater than this value, then the reader may cause a frequency hop before the retried command.

15.15 MODEM.PROTOCOL.FLEX

modem.protocol.flex.read

Read a FLEX tag.

Class	`FUNCTION`	
Parameters	**antenna** :	`INT`
Response	`COMPOUND`	
Permissions	guest	`x`
	admin	`x`

- antenna (optional) - Indicates antenna on which to execute the function. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name, followed by the data read from the tag.

15.16 MODEM.PROTOCOL.FLEX.CONTROL

modem.protocol.flex.control.tx_atten

Transmit attenuation for Flex in ddB.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`300`
Permissions	guest	`r`
	admin	`rw`

This specifies the transmit attenuation applied during Flex transmissions. The value is in ddB so a value of 200 would be an attenuation of 20 dB.

15.17 MODEM.PROTOCOL.FLEX.PHYSICAL

modem.protocol.flex.physical.pilot_threshold

Pilot tone detect threshold in dBm

Class	`VAR`	
Type	`int`	
	default value:	`-65`
	min	`-100`
	max	`0`
Permissions	guest	`-`
	admin	`rw`

Pilot tone detect threshold. If set to 0 (default), the modem will automatically set the threshold based on the noise environment and signal bandwidth.

15.18 MODEM.PROTOCOL.ISO10374.CONTROL

modem.protocol.iso10374.control.dwell_time

The time (millisecs) per round that the reader searches for ISO10374 tags.

Class	`VAR`	
Type	`int`	
	default value:	`100`
	min	`25`
	max	`1000`
Permissions	guest	`r`
	admin	`rw`

The time (millisecs) per round that the reader searches for ISO10374 tags .

modem.protocol.iso10374.control.frame_size

ISO10374 half-frame vs

full frame support.

Class	`VAR`	
Type	`enum`	
	default value: `FULL_FRAME`	
Permissions	guest	`r`
	admin	`rw`

This variable enables support for ISO10374 half-frame tags. For better performance, please select either `FULL_FRAME` or `HALF_FRAME` when the expected tag population is all full or half frame. Only use `FULL_AND_HALF_FRAME` when there is an actual need to read both types of tags.

modem.protocol.iso10374.control.max_reads_per_dwell

The maximum number of tag reads per ISO10374 dwell time.

Class	`VAR`	
Type	`int`	
	default value:	`1`
	min	`0`
	max	`300`
Permissions	guest	`r`
	admin	`rw`

This variable sets the maximum number of reads per ISO10374 dwell time. If set to 0, the reader will continue reporting valid tags found for the entire dwell time. Otherwise, once the reader reaches this number of valid reads during a dwell, it will stop searching for ISO10374 tags and end the dwell period.

modem.protocol.iso10374.control.min_dwell_time

The minimum time (millisecs) per round that the reader searches for iso10374 tags.

Class	`VAR`	
Type	`int`	
	default value:	`5`
	min	`0`
	max	`1000`
Permissions	guest	`r`
	admin	`rw`

The minimum time (millisecs) per round that the reader searches for iso10374 tags. If none are received within that time the modem moves on to the next antenna and/or protocol. If iso10374 tags are received, then the receiver remains listening for the full time specified by `modem.protocol.iso10374.control.dwell_time`. If the `min_dwell_time` variable is set to zero, then the receiver always listens for the full duration.

15 MODEM NAMESPACE

modem.protocol.iso10374.control.min_off_time

The minimum time (millisecs) that RF is off prior to an ISO10374 dwell

If set to 0, the RF off time will be greater than 0, but less than 1 ms.

Class	`VAR`	
Type	`int`	
	default value:	`1`
	min	`0`
	max	`50`
Permissions	guest	`r`
	admin	`rw`

modem.protocol.iso10374.control.report_as_raw_hex

Display tag report as hex data

Class	`VAR`	
Type	`bool`	
	default value: `false`	
Permissions	guest	`r`
	admin	`rw`

This variable enables displaying of ISO10374 tag reports as hex data. When disabled, ISO10374 tag reports are displayed in 6-bit ASCII format.

modem.protocol.iso10374.control.report_tag_immediately

Tag is reported immediately without debouncing if both tag checksums are valid

Debouncing still occurs on tags with partial checksum success.

Class	`VAR`	
Type	`bool`	
	default value: `false`	
Permissions	guest	`r`
	admin	`rw`

modem.protocol.iso10374.control.tx_atten

Transmit attenuation for ISO10374 in ddBm.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`300`
Permissions	guest	`r`
	admin	`rw`

This specifies the transmit attenuation applied during ISO10374 transmissions. The value is in ddBm so a value of 200 would be an attenuation of 20 dBm.

15.19 MODEM.PROTOCOL.ISO10374.FILTER.<N>

modem.protocol.iso10374.filter.<n>.enable

Enable ISO10374 Protocol filtering.

Class	`VAR`	
Type	`bool`	
	default value: `False`	
Permissions	guest	`-`
	admin	`rw`

This variable enables the ISO10374 protocol filtering. When set to `true`, the reader can only report ISO10374 tags with id matching the filtering.

modem.protocol.iso10374.filter.<n>.pattern

Select ISO10374 tag id filtering pattern.

Class	`VAR`	
Type	`string`	
	default value: `"AAAANNNNNNNN"`	
Permissions	guest	–
	admin	`rw`

This variable selects the pattern to filter out the tag ids when 6 bit ASCII is used in the ISO10374 tag id reporting. The first 12 characters of the tag id have to match the pattern in order to be reported. Therre are 3 options: "A" stands for all alphabet characters and "."; "N" stands for numbers from 0 to 9; "X" stands for don’t care.

The following example sets the filter to be starting with 4 alphabet characters or "." and followed by 8 numbers.

```
>>> modem.protocol.iso10374.filter.1.pattern
=AAAANNNNNNNN
ok
```

15.20 MODEM.PROTOCOL.ISO10374.FILTERING

modem.protocol.iso10374.filtering.enable

Enable ISO10374 Protocol filtering.

Class	`VAR`	
Type	`bool`	
	default value: `False`	
Permissions	guest	–
	admin	`rw`

This variable enables the ISO10374 protocol filtering. When set to `true`, the reader can only report ISO10374 tags with id matching the filtering.

15.21 MODEM.PROTOCOL.ISO10374.PHYSICAL

modem.protocol.iso10374.physical.collision_detection_threshold

Configures ISO10374 collision detection threshold.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`256`
Permissions	guest	`r`
	admin	`rw`

Sets ISO10374 collision detection threshold. Higher values mean detection is less likely, lower values mean detection more likely.

modem.protocol.iso10374.physical.fixed_lna

Set fixed value for LNA in ISO10374 mode.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`-1`
	max	`1`
Permissions	guest	–
	admin	`rw`

Force LNA on/off in ISO10374 mode. When 0, LNA forced off for ISO10374. When 1, LNA forced on. When -1, LNA set based on AGC.

15 MODEM NAMESPACE

modem.protocol.iso10374.physical.fixed_vga_dac

Set fixed value for VGA DAC in ISO10374 mode.

Class	VAR	
Type	int	
	default value:	1
	min	-1
	max	2
Permissions	guest	–
	admin	rw

Force VGA to low/med/high gain in ISO10374 mode. When 0, VGA DAC forced to 0. When 1, VGA DAC forced to 512. When 2, VGA DAC forced to 1023. When -1, VGA DAC based on AGC.

modem.protocol.iso10374.physical.pilot_threshold

Pilot tone detect threshold in dBm

Class	VAR	
Type	int	
	default value:	-65
	min	-100
	max	0
Permissions	guest	–
	admin	rw

Pilot tone detect threshold. If set to 0 (default), the modem will automatically set the threshold based on the noise environment and signal bandwidth.

15.22 MODEM.PROTOCOL.ISOB

modem.protocol.isob.lock

Lock specified byte of an ISO-B tag.

Class	FUNCTION		
Parameters	**tag_id**	:	ARRAY
	address	:	INT
	antenna	:	INT
Response	BOOL		
Permissions	guest	x	
	admin	x	

This function locks the specified byte of an ISO-B tag in the field. The parameters supported for this function are:

- `tag_id` - Indicates the id of the tag to read. The tag is first singulated and put in a state for a read operation.
- address - Specifies the address (byte offset) of the byte to be locked on the tag.
- antenna (optional) - Indicates antenna on which to execute the function. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name.

The following example locks the byte at address 202 on a tag with id = `0x3011223344556677788`:

```
>>> modem.protocol.isob.lock (tag_id =
0x3011223344556677788, address = 202)
ok
```

modem.protocol.isob.query_lock

Determine if specified byte of an ISO-B tag is locked.

Class	FUNCTION		
Parameters	**tag_id**	:	ARRAY
	address	:	INT
	antenna	:	INT
Response	COMPOUND		
Permissions	guest	x	
	admin	x	

This function determines if the specified byte of an ISO-B tag in the field is locked. The parameters supported by this function are:

- `tag_id` - Indicates the id of the tag to query. The tag is first singulated and put in a state for a read operation.
- address - Specifies address (byte offset) of byte to query for lock state.
- antenna (optional) - Indicates antenna on which to execute the function. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name followed by the lock status of the byte.

The following example gets the lock ststus of byte at address 202 on a tag with id = `0x3011223344556677788`:

```
>>> modem.protocol.isob.query_lock (tag_id =
0x3011223344556677788, address = 202)
ok byte = locked
```

modem.protocol.isob.read

Read specified data from an ISO-B tag.

Class	`FUNCTION`		
Parameters	**tag_id**	:	`ARRAY`
	address	:	`INT`
	block_count	:	`INT`
	antenna	:	`INT`
Response	`COMPOUND`		
Permissions	guest	`x`	
	admin	`x`	

This function causes the modem to read specified data from an ISO-B tag in the field. This function reads 8 bytes, starting at the address specified from the tag. The parameters supported by this function are:

- `tag_id` - Indicates the id of the tag to be read. The tag is first singulated and put in a state for a read operation.
- address - Specifies the start address of the data to read from the tag.
- `block_count` - Specifies the number of 8 byte blocks to read from the tag.
- antenna (optional) - Indicates antenna on which to execute the function. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name, followed by the data read from the tag.

The following example reads eight bytes of data at address 2 from a tag with id = `0x3011223344556677 88`:

modem.protocol.isob.setup

Shortcut for configuring reader to read `isob_40k` or standard isob tags.

Class	`FUNCTION`		
Parameters	**mode**	:	`ENUM DEFINITIONS.ENUM.PROTOCOL.ISOB.SETUP_MODES`
Response	``		
Permissions	guest	–	
	admin	x	

This function is used to configure the ISOB physical and filter parameters to read either `ISOB_40K` style AVI tags or standard ISOB tags. It affects the following variables:
`modem.protocol.isob.physical.modulation_depth`
`modem.protocol.isob.filtering.enable`
`modem.protocol.isob.filter.1.\*`

modem.protocol.isob.write

Write specified data to an ISO-B tag.

Class	`FUNCTION`		
Parameters	**tag_id**	:	`ARRAY`
	address	:	`INT`
	data	:	`ARRAY`
	antenna	:	`INT`
Response	`COMPOUND`		
Permissions	guest	x	
	admin	x	

This function writes specified data to an ISO-B tag in the field. It writes the supplied data, to the specified address. The parameters supported by this function are:

- `tag_id` - Indicates the ID of the tag to read. The tag is first singulated and put in a state for a read operation.
- address - Specifies the address (byte offset) of the data to write to tag.
- data - specifies the data to be written to the tag starting at address. Length of data must be a byte multiple.

- antenna (optional) - Indicates antenna on which to execute the function. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name. If the byte is locked, the function will fail with a `error.tag.tag_not_writable` response.

The following example writes the byte `0xe4` at address 23 on a tag with ID = `0x3011223344556677 88`:

```
>>> modem.protocol.isob.write (tag_id =
0x3011223344556677 88, data = 0xe4, address =
23)
ok
```

15.23 MODEM.PROTOCOL.ISOB.CONTROL

modem.protocol.isob.control.auto_quiet

Enable auto quiet feature for ISO-B tags.

Class	`VAR`	
Type	`bool`	
	default value: `False`	
Permissions	guest	`r`
	admin	`rw`

This variable enables the auto quiet feature for ISO-B tags. When set to `True`, tags that have been inventoried will remain quiet after having been identified. When set to `False`, tags are reset for every inventory cycle.

MODEM NAMESPACE

modem.protocol.isob.control.cmd_retries

Number of command retries for failed command.

Class	VAR	
Type	int	
	default value:	3
	min	0
	max	8
Permissions	guest	–
	admin	rw

This variable sets the number of command retries for a failed command. When a command sequence fails, it can be retried for the number of times indicated by this parameter. This is transparent to the user.

modem.protocol.isob.control.max_collisions

Maximum number of consecutive collisions before early inventory termination.

Class	VAR	
Type	int	
	default value:	3
	min	1
	max	1500
Permissions	guest	r
	admin	rw

This specifies the maximum number of consecutive collisions before early inventory termination. For noisy environments it prevents extended inventory round time.

modem.protocol.isob.control.multi_tag_support

Support multiple ISOB tags.

Class	`VAR`	
Type	`bool`	
	default value: `False`	
Permissions	guest	`r`
	admin	`rw`

If `true`, multiple ISOB tags can be singulated, assuming the tag correctly implements collision mitigation behavior. If `false`, the following restrictions are imposed when `report_tag_on_select` is `false`:

1 valid response must be seen from the `group_select` command in order to continue the round and

2 the Tag ID received in response to the group select command must match the ID received from the Fail command.

modem.protocol.isob.control.read_variable_cmd

Enable use of optional ISO-B `read_variable` command.

Class	`VAR`	
Type	`bool`	
	default value: `False`	
Permissions	guest	`r`
	admin	`rw`

This variable enables the use of the optional ISO-B `read_variable` command. Any read operations will use the `read_variable` command instead of the read command, which should improve read performance of large amounts of data for those tags that support the `read_variable` command.

modem.protocol.isob.control.report_tag_as_6bitascii

Reports the tags 6 bit ASCII `tag_id`.

Class	`VAR`	
Type	`bool`	
	default value: `True`	
Permissions	guest	`r`
	admin	`rw`

This variable enables sending of an ISO-B tag report with the `tag_id` field containing the tags 6 bit ASCII tag id.

modem.protocol.isob.control.report_tag_on_select

Send an `event.tag.report` on ISO-B tag response to a select command.

Class	`VAR`	
Type	`bool`	
	default value: `True`	
Permissions	guest	`r`
	admin	`rw`

This variable enables sending of a tag report on a successful demodulation of a tag response to one of the ISO-B select commands. The current ISO-B inventory process is terminated with no further commands sent. This would typically be enabled for applications expecting a single ISO-B tag in the field since the normal inventory algorithm is aborted. The select commands sent by the reader are controlled by `modem.protocol.isob.filter.x` variables. When filtering is disabled the reader sends a `GROUP_SELECT_EQ_FLAGS` command targeting the data exchange status bit (`DE_SB`) flag.

modem.protocol.isob.control.tx_atten

Transmit attenuation for ISO-B in ddB.

Class	VAR	
Type	int	
	default value:	0
	min	0
	max	300
Permissions	guest	r
	admin	rw

This specifies the transmit attenuation applied during ISO-B transmissions. The value is in ddB so a value of 200 would be an attenuation of 20 dB.

modem.protocol.isob.control.user_data_addr

The starting address of ISO-B user data.

Class	VAR	
Type	int	
	default value:	24
	min	0
	max	255
Permissions	guest	r
	admin	rw

This specifies the starting address of user data on the ISO-B tag. This is a global variable applied to all ISO-B tags, and is used by functions such as `tag.read_user_data()` and `tag.write_user_data()`.

modem.protocol.isob.control.user_data_length

The block length of ISO-B user data specified in 8 byte blocks.

Class	`VAR`	
Type	`int`	
	default value:	`4`
	min	`0`
	max	`128`
Permissions	guest	`r`
	admin	`rw`

This specifies the block length of ISO-B user data in 8 byte blocks. For ex., if the tag user data length is 32 bytes then, `user_data_length` should be set to 4. This is a global variable applied to all ISO-B tags, and is used by functions such as `tag.read_user_data()` and `tag.write_user_data()`.

15.24 MODEM.PROTOCOL.ISOB.CONTROL.UCODE_EPC_MODE

modem.protocol.isob.control.ucode_epc_mode.enable

Enable EPC format support on UCODE ISO-B tag.

Class	`VAR`	
Type	`bool`	
	default value: `False`	
Permissions	guest	`r`
	admin	`rw`

This variable enables support for the EPC format on UCODE ISO-B tags. NXP UCODE IS0-B tags have a memory configuration that allows for compatibility with EPC 64 and 96 bit formats. When enabled, this variable displays the tag memory in the EPC format. When not enabled, an inventory of the tag will display the memory as stored on the tag.

modem.protocol.isob.control.ucode_epc_mode.length

Set EPC length for EPC enabled UCODE tags.

Class	`VAR`	
Type	`enum`	
	default value: `UCODE_LEN_64`	
Permissions	guest	`r`
	admin	`rw`

This variable sets EPC length (64 or 96 bit) for UCODE tags enabled for EPC mode. The EPC length will determine what tag ID is displayed on inventory.

15.25 MODEM.PROTOCOL.ISOB.FILTER.<N>

modem.protocol.isob.filter.<n>.address

Start address for Select memory comparison.

Class	`VAR`	
Type	`int`	
	default value:	`10`
	min	`0`
	max	`255`
Permissions	guest	`–`
	admin	`rw`

This variable sets the Start address for Select memory comparison. When a `SELECT_EQ`, `SELECT_NE`, `SELECT_GT`, or `SELECT_LT` command is used, this variable contains the address on the tag for where the comparison should start. Not used when FLAGS are compared. Up to 8 bytes starting at address will be compared.

modem.protocol.isob.filter.<n>.data

Data bytes to compare during Select.

Class	VAR	
Type	array	
	default value: 0x0000010000410000	
Permissions	guest	-
	admin	rw

This variable contains the data bytes used in comparison for Select command. These are the data bytes that a tag compares against it's memory location that begins at address. Only tags that match the bytes will respond in an an inventory round. A maximum of 8 bytes can be used in the comparison. A byte comparison is enabled by setting the corresponding bit in mask. For FLAGS Select, the first data byte (MSByte) is used for comparison against the flag data.

modem.protocol.isob.filter.<n>.enable

Enables the ISO-B mask filter.

Class	VAR	
Type	bool	
	default value: True	
Permissions	guest	-
	admin	rw

This variable enables a specific ISO-B mask filter. When set to `true`, only those ISO-B tags that match the filter mask will respond. The filter mask may be a bit pattern which starts at a specific address in the tag.

modem.protocol.isob.filter.<n>.mask

Mask of which bytes to compare during Select.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`255`
Permissions	guest	`-`
	admin	`rw`

This variable selects from 0 to 8 bytes to be compared during the Select process. When a `SELECT_EQ`, `SELECT_NE`, `SELECT_GT`, or `SELECT_LT` command is used, this parameter is a bit mask that determines which bytes are compared. A set bit means compare the byte in `filter.x.data` with tag memory. If a FLAGS Select is used, then a set bit means compare the bit from the first byte of mask with the corresponding FLAG. Some usage for data comparison: mask = `0x00` compare no data mask = `0x80` compare MSByte of the eight bytes from `filter.1.data` mask = `0x40` compare next MSByte of the eight bytes from `filter.1.data` mask = `0x81` compare MSByte and LSByte of the eight bytes from `filter.1.data` mask = `0xff` compare all of the eight bytes from `filter.1.data` Some usage for flag comparison: mask = `0x00` compare no flags mask = `0x80` compare msb of tags FLAG byte with msb of `filter.1.data` MSBByte mask = `0x40` compare next msb of tags FLAG byte with next msb of `filter.1.data` MSBByte mask = `0xff` compare all bits of tags FLAG byte with the MSByte of `filter.1.data` The following example sets the reader to inventory only those tags who have a value of 0x3344 at offset 0x20 in tag memory. The mask of 0xC0 indicates to only compare the 2 MSBytes of the data at address: modem.protocol.isob.filter.1.opcode = SELECT_EQ modem.protocol.isob.filter.1.address = 0x20 modem.protocol.isob.filter.1.mask = 0xC0 modem.protocol.isob.filter.1.data = 0x3344000000000000 modem.protocol.isob.filtering.enable = true The following example sets the reader to inventory only those tags whose DE_SB Flag bit is set: modem.protocol.isob.filter.1.opcode = SELECT_EQ_FLAGS modem.protocol.isob.filter.1.address = 0x00 modem.protocol.isob.filter.1.mask = 0x01 modem.protocol.isob.filter.1.data = 0x01 modem.protocol.isob.filtering.enable = true

modem.protocol.isob.filter.<n>.opcode

Select command used to inventory tags.

Class	`VAR`	
Type	`enum`	
	default value: `SELECT_EQ`	
Permissions	guest	`–`
	admin	`rw`

This variable selects the command to use for tag inventory. The ISO-B protocol uses various Select commands to determine which tags will respond during an Inventory. Tags whose memory matches the data pattern (EQ), does not equal (NE), is greater than (GT) or less than (LT) can be selected. Or the FLAGS field of the tag can be used for comparison (`EQ_FLAGS`, `NE_FLAGS`).

15.26 MODEM.PROTOCOL.ISOB.FILTERING

modem.protocol.isob.filtering.enable

Enables the ISO-B mask filter.

Class	`VAR`	
Type	`bool`	
	default value: `True`	
Permissions	guest	`–`
	admin	`rw`

This variable enables the ISO-B mask filter. When set to `true`, only those ISO-B tags that match the filter mask will respond. The filter mask may be a bit pattern which starts at a specific address in the tag. Up to 8 filters can be set.

15.27 MODEM.PROTOCOL.ISOB.PHYSICAL

modem.protocol.isob.physical.fixed_lna

Set fixed value for LNA in ISOB mode.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`-1`
	max	`1`
Permissions	guest	–
	admin	`rw`

Force LNA on/off in ISOB mode. When 0, LNA forced off for ISOB. When 1, LNA forced on. When -1, LNA set based on AGC.

modem.protocol.isob.physical.fixed_vga_dac

Set fixed value for VGA DAC in ISOB mode.

Class	`VAR`	
Type	`int`	
	default value:	`1`
	min	`-1`
	max	`2`
Permissions	guest	–
	admin	`rw`

Force VGA to low/med/high gain in ISOB mode. When 0, VGA DAC forced to 0. When 1, VGA DAC forced to 512. When 2, VGA DAC forced to 1023. When -1, VGA DAC based on AGC.

modem.protocol.isob.physical.modulation_depth

Modulation depth percentage for ISO-B.

Class	VAR	
Type	int	
	default value:	100
	min	0
	max	100
Permissions	guest	r
	admin	rw

This variable specifies the percent modulation depth for ISO-B transmissions.

modem.protocol.isob.physical.pilot_threshold

Pilot tone detect threshold in dBm

Class	VAR	
Type	int	
	default value:	-70
	min	-100
	max	0
Permissions	guest	-
	admin	rw

Pilot tone detect threshold. If set to 0 (default), the modem will automatically set the threshold based on the noise environment and signal bandwidth.

modem.protocol.isob.physical.return_link_freq

The return link frequency from tag to reader.

Class	`VAR`	
Type	`enum`	
	default value: `LF40`	
Permissions	guest	`r`
	admin	`rw`

This variable sets the return link frequency for the tag to reader communication.

15.28 MODEM.PROTOCOL.ISOB_80K

modem.protocol.isob_80k.lock

Lock specified byte of an ISO-B tag (80k link speeds).

Class	`FUNCTION`		
Parameters	**tag_id**	:	`ARRAY`
	address	:	`INT`
	antenna	:	`INT`
Response	`BOOL`		
Permissions	guest	`x`	
	admin	`x`	

This function locks the specified byte of an ISO-B tag in the field. The parameters supported for this function are:

- `tag_id` - Indicates the id of the tag to read. The tag is first singulated and put in a state for a read operation.
- address - Specifies the address (byte offset) of the byte to be locked on the tag.
- antenna (optional) - Indicates antenna on which to execute the function. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name.

The following example locks the byte at address 202 on a tag with id =

```
0x3011223344556677788:
    >>> modem.protocol.isob_80k.lock (tag_id =
    0x3011223344556677788, address = 202)
    ok
```

modem.protocol.isob_80k.query_lock

Determine if specified byte of an ISO-B tag is locked (80k link speeds).

Class	FUNCTION		
Parameters	**tag_id**	:	ARRAY
	address	:	INT
	antenna	:	INT
Response	COMPOUND		
Permissions	guest	x	
	admin	x	

This function determines if the specified byte of an ISO-B tag in the field is locked. The parameters supported by this function are:

- `tag_id` - Indicates the id of the tag to query. The tag is first singulated and put in a state for a read operation.
- address - Specifies address (byte offset) of byte to query for lock state.
- antenna (optional) - Indicates antenna on which to execute the function. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name followed by the lock status of the byte.

The following example gets the lock ststus of byte at address 202 on a tag with id = `0x3011223344556677788`:

```
    >>> modem.protocol.isob_80k.query_lock
    (tag_id =
    0x3011223344556677788, address = 202)
    ok byte = locked
```

modem.protocol.isob_80k.read

Read specified data from an ISO-B tag (80k link speeds).

Class	`FUNCTION`		
Parameters	**tag_id**	:	`ARRAY`
	address	:	`INT`
	block_count	:	`INT`
	antenna	:	`INT`
Response	`COMPOUND`		
Permissions	guest	`x`	
	admin	`x`	

This function causes the modem to read specified data from an ISO-B tag in the field. This function reads 8 bytes, starting at the address specified from the tag. The parameters supported by this function are:

- `tag_id` - Indicates the id of the tag to be read. The tag is first singulated and put in a state for a read operation.
- address - Specifies the start address of the data to read from the tag.
- `block_count` - Specifies the number of 8 byte blocks to read from the tag.
- antenna (optional) - Indicates antenna on which to execute the function. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name, followed by the data read from the tag.

The following example reads eight bytes of data at address 2 from a tag with id = `0x3011223344556677788`:

modem.protocol.isob_80k.write

Write specified data to an ISO-B tag (80k link speeds).

Class	FUNCTION		
Parameters	**tag_id**	:	ARRAY
	address	:	INT
	data	:	ARRAY
	antenna	:	INT
Response	COMPOUND		
Permissions	guest	x	
	admin	x	

This function writes specified data to an ISO-B tag in the field. It writes the supplied data, to the specified address. The parameters supported by this function are:

- `tag_id` - Indicates the ID of the tag to read. The tag is first singulated and put in a state for a read operation.
- address - Specifies the address (byte offset) of the data to write to tag.
- data - specifies the data to be written to the tag starting at address. Length of data must be a byte multiple.
- antenna (optional) - Indicates antenna on which to execute the function. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name. If the byte is locked, the function will fail with a `error.tag.tag_not_writable` response.

The following example writes the byte `0xe4` at address 23 on a tag with ID = `0x3011223344556677788`:

```
>>> modem.protocol.isob_80k.write (tag_id =
0x3011223344556677788, data = 0xe4, address =
23)
ok
```

15.29 MODEM.PROTOCOL.ISOB_80K.CONTROL

modem.protocol.isob_80k.control.auto_mac

Enables auto MAC control for `ISOB_80K` transactions.

Class	`VAR`	
Type	`bool`	
	default value: `False`	
Permissions	guest	`r`
	admin	`rw`

If `true`, this variable allows the reader to automatically adjust MAC layer settings for optimum performance. When `true`, this variable overrides the settings of the following variables (`i.e.` they no longer apply): - `modem.protocol.isob_80k.control.ucode_epc_mode.enable` (will be disabled) - `modem.protocol.isob_80k.control.report_tag_as_6bitASCII` (will be disabled) - `modem.protocol.isob_80k.control.multi_tag_support` - `modem.protocol.isob_80k.control.single_report` - `modem.protocol.isob_80k.control.report_tag_on_select`

modem.protocol.isob_80k.control.auto_quiet

Enable auto quiet feature for ISO-B tags, when running at 80k link speeds

Class	`VAR`	
Type	`bool`	
	default value: `False`	
Permissions	guest	`r`
	admin	`rw`

This variable enables the auto quiet feature for ISO-B tags. When set to `True`, tags that have been inventoried will remain quiet after having been identified. When set to `False`, tags are reset for every inventory cycle.

modem.protocol.isob_80k.control.cmd_retries

Number of command retries for failed command, when running at 80k link speeds

Class	`VAR`	
Type	`int`	
	default value:	`3`
	min	`0`
	max	`8`
Permissions	guest	–
	admin	`rw`

This variable sets the number of command retries for a failed command. When a command sequence fails, it can be retried for the number of times indicated by this parameter. This is transparent to the user.

modem.protocol.isob_80k.control.max_collisions

Maximum number of consecutive collisions before early inventory termination.

Class	`VAR`	
Type	`int`	
	default value:	`3`
	min	`1`
	max	`1500`
Permissions	guest	`r`
	admin	`rw`

This specifies the maximum number of consecutive collisions before early inventory termination. For noisy environments it prevents extended inventory round time.

MODEM NAMESPACE

modem.protocol.isob_80k.control.multi_tag_support

Support multiple `ISOB_80K` tags.

Class	`VAR`	
Type	`bool`	
	default value: `False`	
Permissions	guest	`r`
	admin	`rw`

If `true`, multiple `ISOB_80K` tags can be singulated, assuming the tag correctly implements collision mitigation behavior. If `false`, the following restrictions are imposed when `report_tag_on_select` is `false`:

1 valid response must be seen from the `group_select` command in order to continue the round and

2 the Tag ID received in response to the group select command must match the ID received from the Fail command.

modem.protocol.isob_80k.control.read_variable_cmd

Enable use of optional ISO-B `read_variable` command, when running at 80k link speeds

Class	`VAR`	
Type	`bool`	
	default value: `False`	
Permissions	guest	`r`
	admin	`rw`

This variable enables the use of the optional ISO-B `read_variable` command. Any read operations will use the `read_variable` command instead of the read command, which should improve read performance of large amounts of data for those tags that support the `read_variable` command.

modem.protocol.isob_80k.control.report_tag_as_6bitascii

Reports the tags 6 bit ASCII `tag_id` (80k link speed).

Class	`VAR`	
Type	`bool`	
	default value: `False`	
Permissions	guest	`r`
	admin	`rw`

This variable enables sending of an ISO-B tag report with the `tag_id` field containing the tags 6 bit ASCII tag id stored in user memory.

modem.protocol.isob_80k.control.report_tag_on_select

Send an `event

tag.report` on ISO-B tag response to a select command, when running at 80k link speeds.

Class	`VAR`	
Type	`bool`	
	default value: `True`	
Permissions	guest	`r`
	admin	`rw`

This variable enables sending of a tag report on a successful demodulation of a tag response to one of the ISO-B select commands. The current ISO-B inventory process is terminated with no further commands sent. This would typically be enabled for applications expecting a single ISO-B tag in the field since the normal inventory algorithm is aborted. The select commands sent by the reader are controlled by `modem.protocol.isob_80k.filter.x` variables. When filtering is disabled the reader sends a `GROUP_SELECT_EQ_FLAGS` command targeting the data exchange status bit (`DE_SB`) flag.

modem.protocol.isob_80k.control.single_report

Send only a single tag report for a tag.

Class	`VAR`	
Type	`bool`	
	default value: `False`	
Permissions	guest	`r`
	admin	`rw`

This variable controls whether one or two tag reports are generated for a tag. When `report_tag_on_select` is `true` then a tag report is generated at that time, and if `single_report` is `true`, the round ends. If `single_report` is `false`, the round continues and another tag report may be generated during the inventory process.

modem.protocol.isob_80k.control.tx_atten

Transmit attenuation for ISO-B in ddBm, when running at 80k link speeds

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`300`
Permissions	guest	`r`
	admin	`rw`

This specifies the transmit attenuation applied during ISO-B transmissions. The value is in ddBm so a value of 200 would be an attenuation of 20 dBm.

modem.protocol.isob_80k.control.user_data_addr

The starting address of ISO-B user data, when running at 80k link speeds

Class	`VAR`	
Type	`int`	
	default value:	`24`
	min	`0`
	max	`255`
Permissions	guest	`r`
	admin	`rw`

This specifies the starting address of user data on the ISO-B tag. This is a global variable applied to all ISO-B tags, and is used by functions such as `tag.read_user_data()` and `tag.write_user_data()`.

modem.protocol.isob_80k.control.user_data_length

The block length of ISO-B user data specified in 8 byte blocks, when running at 80k link speeds

Class	`VAR`	
Type	`int`	
	default value:	`4`
	min	`0`
	max	`128`
Permissions	guest	`r`
	admin	`rw`

This specifies the block length of ISO-B user data in 8 byte blocks. For ex., if the tag user data length is 32 bytes then, `user_data_length` should be set to 4. This is a global variable applied to all ISO-B tags, and is used by functions such as `tag.read_user_data()` and `tag.write_user_data()`.

15.30 MODEM.PROTOCOL.ISOB_80K.FILTER.<N>

modem.protocol.isob_80k.filter.<n>.address

Start address for Select memory comparison when running ISO-B at 80k link speeds.

Class	VAR	
Type	int	
	default value:	10
	min	0
	max	255
Permissions	guest	–
	admin	rw

This variable sets the Start address for Select memory comparison. When a `SELECT_EQ`, `SELECT_NE`, `SELECT_GT`, or `SELECT_LT` command is used, this variable contains the address on the tag for where the comparison should start. Not used when FLAGS are compared. Up to 8 bytes starting at address will be compared.

modem.protocol.isob_80k.filter.<n>.data

Data bytes to compare during Select when running ISO-B at 80k link speeds.

Class	VAR	
Type	array	
	default value:	0x0000010000410000
Permissions	guest	–
	admin	rw

This variable contains the data bytes used in comparison for Select command. These are the data bytes that a tag compares against it's memory location that begins at address. Only tags that match the bytes will respond in an an inventory round. A maximum of 8 bytes can be used in the comparison. A byte comparison is enabled by setting the corresponding bit in mask. For FLAGS Select, the first data byte (MSByte) is used for comparison against the flag data.

modem.protocol.isob_80k.filter.<n>.enable

Enables the ISO-B (80k link speed) mask filter.

Class	VAR	
Type	bool	
	default value: True	
Permissions	guest	–
	admin	rw

This variable enables a specific ISO-B mask filter when running at 80k link speeds. When set to `true`, only those ISO-B tags that match the filter mask will respond. The filter mask may be a bit pattern which starts at a specific address in the tag.

modem.protocol.isob_80k.filter.<n>.mask

Mask of which bytes to compare during Select when running ISO-B at 80k link speeds.

Class	VAR	
Type	int	
	default value:	0
	min	0
	max	255
Permissions	guest	–
	admin	rw

This variable selects from 0 to 8 bytes to be compared during the Select process. When a `SELECT_EQ`, `SELECT_NE`, `SELECT_GT`, or `SELECT_LT` command is used, this parameter is a bit mask that determines which bytes are compared. A set bit means compare the byte in `filter.x.data` with tag memory. If a FLAGS Select is used, then a set bit means compare the bit from the first byte of mask with the corresponding FLAG. Some usage for data comparison: mask = `0x00` compare no data mask = `0x80` compare MSByte of the eight bytes from `filter.1.data` mask = `0x40` compare next MSByte of the eight bytes from `filter.1.data` mask = `0x81` compare MSByte and LSByte of the eight bytes from `filter.1.data` mask = `0xff` compare all of the eight bytes from `filter.1.data` Some usage for flag comparison: mask = `0x00` compare no flags mask = `0x80` compare msb of tags FLAG byte with msb of `filter.1.data` MSBByte mask = `0x40` compare next msb of tags FLAG byte with next msb of `filter.1.data` MSBByte mask = `0xff` compare all bits of tags FLAG byte with the MSByte of `filter.1.data` The following example sets the reader to inventory only those tags who have a value of 0x3344 at offset 0x20 in tag memory. The mask of 0xC0 indicates to only compare the 2 MSBytes of the data at address: modem.protocol.isob_80k.filter.1.opcode = SELECT_EQ modem.protocol.isob_80k.filter.1.address = 0x20 modem.protocol.isob_80k.filter.1.mask = 0xC0 modem.protocol.isob_80k.filter.1.data = 0x3344000000000000 modem.protocol.isob_80k.filtering.enable = true The following example sets the reader to inventory only those tags whose DE_SB Flag bit is set: modem.protocol.isob_80k.filter.1.opcode = SELECT_EQ_FLAGS modem.protocol.isob_80k.filter.1.address = 0x00 modem.protocol.isob_80k.filter.1.mask = 0x01 modem.protocol.isob_80k.filter.1.data = 0x01 modem.protocol.isob_80k.filtering.enable = true

modem.protocol.isob_80k.filter.<n>.opcode

Select command used to inventory tags, when running ISO-B with 80k link speeds

Class	`VAR`	
Type	`enum`	
	default value: `SELECT_EQ`	
Permissions	guest	–
	admin	`rw`

This variable selects the command to use for tag inventory. The ISO-B protocol uses various Select commands to determine which tags will respond during an Inventory. Tags whose memory matches the data pattern (EQ), does not equal (NE), is greater than (GT) or less than (LT) can be selected. Or the FLAGS field of the tag can be used for comparison (`EQ_FLAGS`, `NE_FLAGS`).

15.31 MODEM.PROTOCOL.ISOB_80K.FILTERING

modem.protocol.isob_80k.filtering.enable

Enables the ISO-B (80k link speed) mask filter.

Class	`VAR`	
Type	`bool`	
	default value: `True`	
Permissions	guest	–
	admin	`rw`

This variable enables the ISO-B mask filter for ISO-B running at 80k link speeds. When set to `true`, only those ISO-B tags that match the filter mask will respond. The filter mask may be a bit pattern which starts at a specific address in the tag. Up to 8 filters can be set.

15.32 MODEM.PROTOCOL.ISOB_80K.PHYSICAL

modem.protocol.isob_80k.physical.fixed_lna

Set fixed value for LNA in `ISOB_80K` mode.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`-1`
	max	`1`
Permissions	guest	–
	admin	`rw`

Force LNA on/off in `ISOB_80K` mode. When 0, LNA forced off for `ISOB_80K.` When 1, LNA forced on. When -1, LNA set based on AGC.

modem.protocol.isob_80k.physical.fixed_vga_dac

Set fixed value for VGA DAC in `ISOB_80K` mode.

Class	`VAR`	
Type	`int`	
	default value:	`1`
	min	`-1`
	max	`2`
Permissions	guest	–
	admin	`rw`

Force VGA to low/med/high gain in `ISOB_80K` mode. When 0, VGA DAC forced to 0. When 1, VGA DAC forced to 512. When 2, VGA DAC forced to 1023. When -1, VGA DAC based on AGC.

modem.protocol.isob_80k.physical.modulation_depth

Modulation depth percentage for ISO-B when running at 80k link speeds.

Class	`VAR`	
Type	`int`	
	default value:	`100`
	min	`0`
	max	`100`
Permissions	guest	`r`
	admin	`r`

This variable specifies the percent modulation depth for ISO-B transmissions.

modem.protocol.isob_80k.physical.pilot_threshold

Pilot tone detect threshold in dBm

Class	VAR	
Type	int	
	default value:	-60
	min	-100
	max	0
Permissions	guest	-
	admin	rw

Pilot tone detect threshold. If set to 0 (default), the modem will automatically set the threshold based on the noise environment and signal bandwidth.

modem.protocol.isob_80k.physical.threshold1

Set threshold1 for `ISOB_80K` mode

Unless otherwise directed by support, leave this value to its default.

Class	VAR	
Type	int	
	default value:	5000
	min	2000
	max	20000
Permissions	guest	-
	admin	rw

15.33 MODEM.PROTOCOL.ISOC

modem.protocol.isoc.block_erase

Erase the specified section of an ISO-C tag.

Class	FUNCTION		
Parameters	**tag_id**	:	ARRAY
	pwd	:	ARRAY
	mem_bank	:	INT
	word_ptr	:	INT
	word_count	:	INT
	antenna	:	INT
Response	BOOL		
Permissions	guest	x	
	admin	x	

This function erases the specified section of an ISO-C tag in the field. The parameters supported for this function are:

- `tag_id` (optional) - Indicates the ID of the tag to erase. If the `tag_id` is not supplied, the first tag found will be erased. The tag is first singulated and put in a state for an erase operation.
- password (optional) - Password used to erase locked portions of a tag.
- `mem_bank` - Specifies which tag memory bank is to be erased (0 = Reserved, 1 = EPC, 2 = TID, 3 = User memory).
- `word_ptr` - Specifies the starting word (16 bit) address in the memory bank for the erase.
- `word_count` - Specifies the number of 16-bit words to erase.
- antenna (optional) - Indicates antenna on which to execute the function. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name.

The following example erases one word at offset 4 from the epc memory section of a tag ID = `0x30112233445566778899aabb`:

```
>>> modem.protocol.isoc.block_erase (tag_id =
0x30112233445566778899aabb, mem_bank = 1,
word_ptr = 4, word_count
= 1)
ok
```

modem.protocol.isoc.block_permalock

Perform BlockPermalock on an ISO-C tag.

Class	`FUNCTION`		
Parameters	**tag_id**	:	`ARRAY`
	pwd	:	`ARRAY`
	mem_bank	:	`INT`
	block_ptr	:	`INT`
	block_range	:	`INT`
	lock	:	`BOOL`
	mask	:	`ARRAY`
	antenna	:	`INT`
Response	`COMPOUND`		
Permissions	guest	`x`	
	admin	`x`	

This function will perform the Gen2 optional command BlockPermalock on an ISO-C tag in the field. Block sizes are vendor defined. The parameters supported by this function are:

- `tag_id` (optional) - Indicates the ID of the tag to operate on. If the `tag_id` is not supplied, the first tag found will be operated on. The tag is first singulated and put in a state for a BlockPermalock operation.
- `pwd` (optional) - Password used to access the tag. .
- `mem_bank` - Specifies which tag memory bank to operate on (0 = Reserved, 1 = EPC, 2 = TID, 3 = User memory).
- `block_ptr` - Specifies the starting block to permalock.
- `block_range` - Specifies the number of blocks to permalock.
- lock - Specifies whether the operation is a read of lock status (`false`) or lock of blocks (`true`)

- mask - Specifies the lock mask applied to the blocks. If a bit in the mask is 1 then the corresponding block will be permalocked. If a bit in the mask is 0 then the current lock status of the block is retained by the tag.
- antenna (optional) - Indicates antenna on which to execute the function. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name.

The following command permalocks 3 blocks of user memory starting at the first block on a tag with ID = `0x30112233445566778899aabb`:

```
>>> modem.protocol.isoc.block_permalock
(tag_id =

0x30112233445566778899aabb,mem_bank=3,block_p
tr=0,block_range=1,lock=true,mask=0xE000)

ok
```

The following command reads the permalock status of user memory starting at the first block on a tag with ID = `0x30112233445566778899aabb`:

```
>>> modem.protocol.isoc.block_permalock
(tag_id =

0x30112233445566778899aabb,mem_bank=3,block_p
tr=0,block_range=1,lock=false)

ok lock_status = 0xE000
```

modem.protocol.isoc.cmd_seq_end

End the command sequence for an ISO-C tag.

Class	`FUNCTION`		
Parameters	**antenna**	:	`INT`
Response	`BOOL`		
Permissions	guest	`x`	
	admin	`x`	

modem.protocol.isoc.cmd_seq_start

Singulate an ISO-C tag and ready it for subsequent commands

This command will find the tag specified by `tag_id` (optional) or the first tag in the field, and get a handle for subsequent commands. The commands sent between `cmd_seq_start` and `cmd_seq_end` will not cause a resingulation of the tag, but will use the handle found by `cmd_seq_start`. If "antenna" is specified (optional), all subsequent commands are issued on "antenna" if `cmd_seq_start` was successful. The "timeout" parameter (optional) is a millisecond value that limits the length of time the modem waits for subsequent commands before returning to normal operation. If "timeout" occurs it has the same effect as the modem receiving `cmd_seq_end` function. The "timeout" parameter defaults to 200 ms if not specified.

Class	`FUNCTION`		
Parameters	**tag_id**	:	`ARRAY`
	antenna	:	`INT`
	timeout	:	`INT`
Response	`BOOL`		
Permissions	guest	`x`	
	admin	`x`	

The following example finds the ISO-C tag with `tag_id` of `0x30112233445566778899aabb`, gets a handle, and then subsequent commands use that handle. If the result is not "ok" then the tag was not found and the subsequent commands will resingulate using their `tag_id` field if specified.

```
>>> modem.protocol.isoc.cmd_seq_start(tag_id
=

0x30112233445566778899aabb)

ok

>>>

modem.protocol.isoc.read(mem_bank=3,word_ptr=
0,word_count=1)

ok data=0x1234

>>>

modem.protocol.isoc.read(mem_bank=3,word_ptr=
1,word_count=1)

ok data=0x5678
```

```
>>> modem.protocol.isoc.cmd_seq_end()
ok
```

modem.protocol.isoc.conf_test

Generate the appropriate ISO-C conformance test commands.

Class	FUNCTION		
Parameters	**command**	:	ENUM DEFINITIONS.ENUM.PROTOCOL.ISOC.TEST_COMMANDS
	data	:	ARRAY
	num_bits	:	INT
	antenna	:	INT
Response	BOOL		
Permissions	guest	–	
	admin	x	

Generate the appropriate ISO-C conformance test commands. Can also be used to send specific bit stream of data. The parameters supported by this function are:

- command (optional) - The Gen2 conformance command sequence.
- data (optional) - The data bits to be sent using ISOC transmission.
- `num_bits` (optional) - The number of bits from the `<data>` parameter to be sent.

```
>>>
modem.protocol.isoc.conf_test(command=QUERY)
>>>
modem.protocol.isoc.conf_test(data=0xaa55,
num_bits=9) This will send the bits
101010100b using the ISOC
protocol.
```

modem.protocol.isoc.lock

Low-level lock command for ISO-C tag.

Class	`FUNCTION`		
Parameters	**tag_id**	:	`ARRAY`
	pwd	:	`ARRAY`
	payload	:	`INT`
	antenna	:	`INT`
Response	`COMPOUND`		
Permissions	guest	`x`	
	admin	`x`	

This function uses the specified payload (20 bit value) to lock an ISO-C tag in the field. The parameters supported by this function are:

- `tag_id` (optional) - Indicates the ID of the tag to lock. If the `tag_id` is not supplied, the first tag found will be locked. The tag is first singulated and put in a state for a lock operation.
- `pwd` (optional) - Password used to lock the tag. .
- payload - Specifies the payload used in the lock command. See ISOC 18000-6C spec for details.
- antenna (optional) - Indicates antenna on which to execute the function. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name.

The following example locks user memory on a tag with ID = `0x30112233445566778899aabb`:

```
>>> modem.protocol.isoc.lock (tag_id =
0x30112233445566778899aabb, payload =
0x00c03)
ok
```

modem.protocol.isoc.read

Read specified data from an ISO-C tag.

Class	`FUNCTION`		
Parameters	**tag_id**	:	`ARRAY`
	pwd	:	`ARRAY`
	mem_bank	:	`INT`
	word_ptr	:	`INT`
	word_count	:	`INT`
	antenna	:	`INT`
Response	`COMPOUND`		
Permissions	guest	`x`	
	admin	`x`	

This function reads specified data from an ISO-C tag in the field. The parameters supported by this function are:

- `tag_id` (optional) - Indicates the ID of the tag to read. If the `tag_id` is not supplied, the first tag found is read. The tag is first singulated and put in a state for a read operation.
- `pwd` (optional) - Password used to read locked portions of a tag.
- `mem_bank` - Specifies which tag memory bank is to read (0 = Reserved, 1 = EPC, 2 = TID, 3 = User memory).
- `word_ptr` - Specifies the starting word (16 bit) address in the memory bank for the read.
- `word_count` - Specifies the number of 16-bit words to read. If the `word_count` = 0 the entire memory bank is read.
- antenna (optional) - Indicates antenna on which to execute the function. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name, followed by the data read from the tag.

The following example reads four words at offset 2 from the user memory section of a tag with ID = `0x30112233445566778899aabb`:

```
>>> modem.protocol.isoc.read (tag_id =
0x30112233445566778899aabb, mem_bank = 3,
word_ptr = 2, word_count
= 4)
ok data = 0x9988776655443322
```

modem.protocol.isoc.recomm

Recommission an ISO-C tag.

Class	FUNCTION		
Parameters	**tag_id**	:	ARRAY
	kill_pwd	:	ARRAY
	recomm	:	INT
	antenna	:	INT
Response	BOOL		
Permissions	guest	x	
	admin	x	

This function recommissions an ISO-C tag in the field. The parameters supported by this function are:

- `tag_id` (optional) - Indicates the ID of the tag to recommission. If the `tag_id` is not supplied, the first tag found is operated on. The tag is first singulated and put in a state for the recommission operation.
- `kill_pwd` indicates the kill password required to perform the operation.
- recomm is an integer value which indicates the recommission operation to perform. The user should have a thorough understanding of the ISO-C specification (Gen2 spec `v1.2`) in regards to tag recommissioning. The recomm parameter will become the 3 bit recommission operation sent to the tag. valid values are: 1, 2, 3, 4, 5, 6, 7
- antenna (optional) - Indicates antenna on which to execute the function. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name.

The following example recommissions a tag with ID = `0x30112233445566778899aabb` and `kill_pwd=12345678.` The recomm value of 2 causes User memory to become unusable:

```
>>> modem.protocol.isoc.recomm (tag_id =
0x30112233445566778899aabb,
kill_pwd=0x12345678, recomm=2)
ok
```

modem.protocol.isoc.write

Write specified data to an ISO-C tag.

Class	FUNCTION		
Parameters	**tag_id**	:	ARRAY
	pwd	:	ARRAY
	mem_bank	:	INT
	word_ptr	:	INT
	data	:	ARRAY
	antenna	:	INT
Response	COMPOUND		
Permissions	guest	x	
	admin	x	

This function writes specified data to an ISO-C tag in the field. The parameters supported by this function are:

- `tag_id` (optional) - Indicates the ID of the tag to write. If the `tag_id` is not supplied, the first tag found will be written. The tag is first singulated and put in a state for a write operation.
- `pwd` (optional) - Password used to write locked portions of a tag. .
- `mem_bank` - Specifies which tag memory bank to write (0 = Reserved, 1 = EPC, 2 = TID, 3 = User memory).
- `word_ptr` - Specifies the starting word (16 bit) address in the memory bank for the write.
- data - Specifies the data to write. If the data is not on word boundaries, it is padded with 0s to make a full word.
- antenna (optional) - Indicates antenna on which to execute the function. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name.

The following example writes `0x12345678` at offset 3 from the EPC memory section of a tag with ID = `0x30112233445566778899aabb`:

```
>>> modem.protocol.isoc.write (tag_id =
0x30112233445566778899aabb, mem_bank = 1,
word_ptr = 3, data =
0x12345678)
ok
```

15.34 MODEM.PROTOCOL.ISOC.ALIEN

modem.protocol.isoc.alien.block_read_lock

Perform Alien custom command, BlockReadLock

Class	FUNCTION		
Parameters	**tag_id**	:	ARRAY
	pwd	:	ARRAY
	antenna	:	INT
	read_lock	:	INT
Response	BOOL		
Permissions	guest	x	
	admin	x	

Perform Alien custom command, BlockReadLock. This command will lock/unlock user memory blocks for reading on some Alien silicon based tags. A block is 4 contiguous words in user memory. Block 0 is words 0-3, Block 1 is words 4-7, etc. The `read_lock` parameter is a bit map with the following definition: MSB LSB bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0 0 1 2 3 4 5 6 7 Block Affected If the bit = 1, then the block is locked. If the bit = 0, then the block is unlocked. This command is only successful when the tag is in the secured state.

The following example locks Block 0, which is the first 4 words of user memory, and unlocks all other Blocks.

```
>>>
modem.protocol.isoc.alien.block_read_lock(pwd
=12345678,
read_lock=0x80)
ok
```

15.35 MODEM.PROTOCOL.ISOC.CONTROL

modem.protocol.isoc.control.cmd_retries

Number of command retries for failed command.

Class	VAR	
Type	int	
	default value:	3
	min	0
	max	8
Permissions	guest	r
	admin	rw

This variable sets the number of command retries for failed command. When a command sequence fails, it can be retried for the number of times indicated by this parameter. This is transparent to the user.

modem.protocol.isoc.control.crc_retries

Number of times tag will be re-ACK'd on bad EPC CRC

Class	VAR	
Type	int	
	default value:	1
	min	0
	max	8
Permissions	guest	r
	admin	rw

This variable sets how many times a tag ACKnowledge will be sent on a bad EPC CRC before moving on in the inventory cycle. If set to zero, the tag will not have it's inventory flag toggled. If the retries are set to one or more and the tag continues to give a bad CRC, the tag will have it's inventory state toggled before reporting the bad CRC.

modem.protocol.isoc.control.display_tag_crc

Display of tag CRC.

Class	`VAR`	
Type	`bool`	
	default value: `False`	
Permissions	guest	`r`
	admin	`rw`

This variable enables display of the tag CRC. When set to `true`, entire tag data contents, including CRC, is displayed. When set to `false`, only the EPC data bits are displayed.

modem.protocol.isoc.control.enable_umi_method_2

Enable support of Method 2 for UMI control.

Class	`VAR`	
Type	`bool`	
	default value: `False`	
Permissions	guest	`r`
	admin	`rw`

This variable enables Method 2 (see ISO-C spec) for controlling the UMI bit of the PC word. If set to `true`, the interrogator will write the UMI bit based on the bit values of bits 3-7 (bit 0 is msb) of the first word of user memory. If any of these bits are set, then the UMI bit of the PC word will be set, otherwise the UMI bit is cleared. In addition, if EPC (UII) memory is locked, then user memory will also be locked, and vice versa. This does not apply to BlockPermalock, in which case the tag controls the UMI bit. If `false`, then the interrogator does not modify the UMI bit or perform synchronous locking.

modem.protocol.isoc.control.include_alllock_bits

Include the permalock bit in lock procedure.

Class	`VAR`	
Type	`bool`	
	default value: `true`	
Permissions	guest	`r`
	admin	`rw`

This variable controls whether both lock bits of a ISO-C lock field are used in the lock procedure. The default setting is to use both lock bits, `pwd`-read/write and permalock, when locking a memory space. When set to `false`, the variable `modem.protocol.isoc.control.include_permalock_bit` gives finer control over the lock bits used.

modem.protocol.isoc.control.include_permalock_bit

Include the permalock bit in lock procedure.

Class	`VAR`	
Type	`bool`	
	default value: `true`	
Permissions	guest	`-`
	admin	`rw`

This variable controls whether the permalock bit is included in the lock procedure when a lock command has been issued. If `false`, only the `pwd`-read/write bit is used. This variable is only used when `modem.protocol.isoc.control.include_alllock_bits` is `false`.

modem.protocol.isoc.control.inventory_both_targets

Enable inventory of tags for both inventoried flag targets.

Class	`VAR`	
Type	`bool`	
	default value: `True`	
Permissions	guest	`r`
	admin	`rw`

This variable enables inventory of tags from A to B, and B back to A. When set to `true`, tags will be inventoried from the B state back to the A state. When `false`, only tags in the A state are inventoried to the B state. This variable is overridden when the `auto_mac.enable` control variable is `true` (default).

modem.protocol.isoc.control.max_incr_slots_q

Max Q = `number_slots_Q` + `max_incr_slots_Q`.

Class	`VAR`	
Type	`int`	
	default value:	`2`
	min	`0`
	max	`15`
Permissions	guest	`r`
	admin	`rw`

This variable controls how large the Q value from the `ISO_C` spec can grow. The Q value is capped at the initial Q value (`number_slots_Q`) plus this `max_incr_slots_Q` value.

modem.protocol.isoc.control.mem_bank_for_selection

Memory bank to which Select command applies.

Class	`VAR`	
Type	`enum`	
	default value: `MEMBANK_EPC`	
Permissions	guest	`r`
	admin	`rw`

This variable is the memory bank that Select command applies to perform filtering.

modem.protocol.isoc.control.number_slots_q

Initial number of slots in inventory round.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`15`
Permissions	guest	`r`
	admin	`rw`

This variable sets the initial number of slots in inventory round. The initial number of slots is the Q value from the ISO-C spec, where the number of slots is equal to 2 to the Q power. This variable is overridden when the `auto_mac.enable` control variable is `true` (default).

To set the number of slots to 1, set `number_slots_Q` = 0. To set the number of slots to 2, set number _`slots_Q` = 1.

modem.protocol.isoc.control.query_sel

ISO-C sel for the Query command.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`4`
Permissions	guest	`r`
	admin	`rw`

This variable sets the ISO-C sel used by the initial Query command. If set to 0, the value used in the Query is predetermined. For finer control this variable allows for specific sel values to be sent: 1 - All tags respond to Query, 2 - All tags respond to Query, 3 - Tags with negated SL flag respond to Query, 4 - Tags with asserted SL flag respond to Query.

modem.protocol.isoc.control.query_session

ISO-C session for the Query command.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`4`
Permissions	guest	`r`
	admin	`rw`

This variable sets the ISO-C session used by the initial Query command. If set to 0, the value used in the Query is `modem.protocol.isoc.control.session_id.` For finer control this variable allows for specific session values to be sent: 1 - Session 0, 2 - Session 1, 3 - Session 2, 4 - Session 3

modem.protocol.isoc.control.query_target

ISO-C target for the Query command.

Class	`VAR`	
Type	`enum`	
	default value: `TGT_A`	
Permissions	guest	`r`
	admin	`rw`

This variable sets the ISO-C target used by the initial Query command. ISO-C tags have an inventoried flag for each session. This flag will change state from from `TGT_A` to `TGT_B` or vice verse when a tag has been singulated.

modem.protocol.isoc.control.select_cmd_period

Set Select command period at start of inventory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`1023`
Permissions	guest	`r`
	admin	`rw`

This variable controls how often a Select command is sent when ISO-C filtering is not enabled. The Select command is sent at the start of every N inventory rounds. If set to 0, the Select command is never sent before inventroy rounds. This variable is overridden when the `auto_mac.enable` control variable is `true` (default).

The following examples set to send a Select command every 1, 5, and 0 (do not send) inventory rounds.

```
modem.protocol.isoc.control.select_cmd_period
=1
```

```
modem.protocol.isoc.control.select_cmd_period
=5
```

```
modem.protocol.isoc.control.select_cmd_period
=0
```

15 MODEM NAMESPACE

modem.protocol.isoc.control.session_id

ISO-C Session to use for tag communication.

Class	`VAR`	
Type	`enum`	
	default value: `SESSION_0`	
Permissions	guest	`r`
	admin	`rw`

This variable sets the ISO-C session to use for tag communication. The ISO-C specification allows for up to four sessions to be used concurrently. A tag stores state information for each session, so that a Reader in one Session does not disrupt the tag state for another Reader using a different Session.

modem.protocol.isoc.control.terminate_threshold

Round termination threshold for ending unproduction inventory rounds

Class	`VAR`	
Type	`int`	
	default value:	`31`
	min	`0`
	max	`1000`
Permissions	guest	`-`
	admin	`rw`

This variable controls how many unsuccessful slots in a row can occur prior to stopping the inventory round.

MODEM NAMESPACE

modem.protocol.isoc.control.tx_atten

Transmit attenuation for ISO-C in ddB.

Class	`VAR`		
Type	`int`		
	default value:	`0`	
	min	`0`	
	max	`300`	
Permissions	guest	`r`	
	admin	`rw`	

This specifies the transmit attenuation applied during ISO-C transmissions. The value is in ddB so a value of 200 would be an attenuation of 20 dB.

modem.protocol.isoc.control.use_block_write

Enable ISO-C block write capability.

Class	`VAR`	
Type	`bool`	
	default value: `False`	
Permissions	guest	`r`
	admin	`rw`

This variable enables ISO-C block write capability. This variable must be supported by tag. When set to `true`, the reader issues the optional ISO-C command BlockWrite for write operations. When set to `false`, the standard 16-bit Write command is issued. The Write command is mandatory and must be supported by all tags.

modem.protocol.isoc.control.user_data_length

The number of 16-bit words of ISO-C `user_memory` data to read during inventory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`255`
Permissions	guest	`r`
	admin	`rw`

This specifies the number of 16-bit words of ISO-C user data to read during inventory or with `tag.read_user_data()`. For ex., if the tag user data length is 512 bits then, `user_data_length` can be set to 32. This is a global variable applied to all ISO-C tags, and is used by functions such as `tag.read_user_data()`. The default value of 0 causes the entire memory bank of user data to be read. Tags that have up to 8192 bits (512 words) of `user_data` can use the default setting of 0. Tags that have more then 8192 bits (512 words) of user memory will require that a non-zero value be used for the `user_data_length` since the reader limits the read result to a maximum of 8192 bits. The ISO-C read command has a maximum word count of 255 words, so that is the maximum number of words that can be read at one time when reading the entire memory bank cannot be done. Tags that have user memory banks larger than 8192 bits require multiple explicit reads to return the entire memory bank. No more than 8192 bits of user data can be displayed during inventory, eg. when `tag.reporting.report_fields=user_data.` This variable should be used in conjunction with `user_data_offset` to specify the portion of `user_memory` to be read. If the offset+length exceeds the tag's memory space then the read will fail.

modem.protocol.isoc.control.user_data_offset

The word offset of ISO-C `user_memory` data at which to start reading/writing.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`INT_MAX`
Permissions	guest	`r`
	admin	`rw`

This specifies the word offset in `user_memory` data on the ISO-C tag at which to start reads for inventory operations or with `tag.read_user_data`, or writes with `tag.write_user_data` (). This is a global variable applied to all ISO-C tags, and is used by functions such as `tag.read_user_data()` and `tag.write_user_data()`. The default value of 0 causes the read/write to start at offset 0 in the `user_memory.` This variable should be used in conjunction with `user_data_length` to specify the portion of `user_memory` to be read. If the offset+length exceeds the tag's memory space then the read will fail.

modem.protocol.isoc.control.verify_write

Enable read verify of ISOC write commands.

Class	`VAR`	
Type	`bool`	
	default value: `True`	
Permissions	guest	`r`
	admin	`rw`

This variable enables read back verification of ISOC writes. When set to `true`, a read command is issued following a write to verify the write success. When set to `false`, no read back is performed.

modem.protocol.isoc.control.write_rssi_advanced_algorithm

Enable the advanced write RSSI algorithm

If `true`, the algorithm is applied before every write attempt when `write_rssi_max_acks` is non-zero. If `false`, then the `write_rssi_threshold` controls whether a write command is issued. The algorithm will end and a write attempted for any of the following: The write RSSI threhold is reached. The tag RSSI values have been increasing, and start to level off. The tag RSSI values have been increasing, and start to decrease. The tag RSSI values are only decreasing.

Class	`VAR`	
Type	`bool`	
	default value: `false`	
Permissions	guest	`r`
	admin	`rw`

modem.protocol.isoc.control.write_rssi_enable

Enable the write RSSI functionality.

Class	`VAR`	
Type	`bool`	
	default value: `false`	
Permissions	guest	`r`
	admin	`rw`

Enable the write RSSI functionality. If `true`, then ISOC writes will be gated by RSSI, either through the `write_rssi_threshold` or the `write_rssi_advanced_algorithm.` If `false`, then ISOC writes are attempted immediately, which is the legacy behavior.

modem.protocol.isoc.control.write_rssi_max_acks

The maximum number of ISOC ACKs tried during write RSSI algorithm

This enables control for write RSSI algorithms, whether simple `write_rssi_threshold` or the `write_rssi_advanced_algorithm.` If 0, then neither the `write_rssi_threshold` or `write_rssi_advanced_algorithm` is attempted and the reader issues the write command immediately. If non-zero, the reader will issue ACK's until the max is reached, or the ending condition is met, ie. thresold reached, or algorithm ends.

Class	`VAR`	
Type	`int`	
	default value:	`64`
	min	`0`
	max	`100`
Permissions	guest	`r`
	admin	`rw`

modem.protocol.isoc.control.write_rssi_threshold

The RSSI (in ddbm) at which an ISOC write command is attempted

If not reached then no ISOC write command is attempted (unless the `write_rssi_advanced_algorithm` is enabled). This is only applied when `modem.protocol.isoc.control.write_rssi_max_acks` is non-zero.

Class	`VAR`	
Type	`int`	
	default value:	`-550`
	min	`-700`
	max	`0`
Permissions	guest	`r`
	admin	`rw`

15.36 MODEM.PROTOCOL.ISOC.FILTER.<N>

modem.protocol.isoc.filter.<n>.action

Action tag takes based on filter match.

Class	`VAR`	
Type	`enum`	
	default value: `ASSERT_DEASSERT`	
Permissions	guest	`–`
	admin	`rw`

This variable is the action a tag takes based on filter match. An ISO-C tag takes action based on whether it matches/unmatches the filter mask. The action applies to the SL flag of the tag if `modem.protocol.isoc.filtering.use_session` is `false`. If `modem.protocol.isoc.filtering.use_session` is `true`, then the action applies to the tags inventoried flag for the session specified in `modem.protocol.isoc.control.session_id.` The enum name indicates the action for a match before the underscore, and the action for an unmatch after the underscore. The SL flag will be asserted, deasserted, negated or not acted on. The session inventoried flag will be set to A (asserted), B (deasserted), flipped A to B or B to A (negated) or not acted on. To inventory only tags that have value `0x22` at offset `0x30`, OR value `0x44` at offset `0x30`, set up the filter variables for 2 filters, and then use action `ASSERT_DEASSERT` for the first filter and action `ASSERT_NOTHING` for the second filter. To inventory only tags that have value `0x22` at offset `0x30` AND value `0x44` at offset `0x40`, set up the filter variables for 2 filters, and then use action `ASSERT_DEASSERT` for the first filter and action `NOTHING_DEASSERT` for the second filter. To inventory only tags that have value `0x22` at offset `0x30`, AND exclude tags with value `0x40` at offset `0x40`, set up the filter variables for 2 filters, and then use action `ASSERT_DEASSERT` for the first filter and action `DEASSERT_NOTHING` for the second filter.

modem.protocol.isoc.filter.<n>.enable

Enables the ISO-C mask filter.

Class	`VAR`	
Type	`bool`	
	default value: `False`	
Permissions	guest	`–`
	admin	`rw`

This variable enables the ISO-C mask filter. When set to `true`, only those ISO-C tags that match the filter mask will respond. The filter mask is a bit pattern which starts at a specific bit offset in the tag and has a specific length.

modem.protocol.isoc.filter.<n>.length

Bit length of the ISO-C mask filter.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`255`
Permissions	guest	`–`
	admin	`rw`

This variable is the bit length of the ISO-C mask filter. This variable specifies the bit length of the ISO-C mask filter.

modem.protocol.isoc.filter.<n>.mask

Bit mask for the ISO-C mask filter.

Class	`VAR`	
Type	`array`	
	default value: `0x00`	
Permissions	guest	`–`
	admin	`rw`

This variable is the bit mask for the ISO-C mask filter. This is the mask that a tag compares against it's memory location that begins at offset and ends length bits later. Only tags that match the mask will respond in a round if filtering is enabled.

modem.protocol.isoc.filter.<n>.mem_bank

Memory bank used for the ISO-C mask filter.

Class	`VAR`	
Type	`enum`	
	default value: `NOT_USED`	
Permissions	guest	–
	admin	`rw`

This variable is the memory bank for the ISO-C mask filter. This is the memory bank that a tag uses when comparing against it's memory location that begins at offset and ends length bits later. Only tags that match the mask will respond in a round if filtering is enabled. If set to `NOT_USED`, then the variable `modem.protocol.isoc.control.mem_bank_for_selection` will control which memory bank is used for this filter.

modem.protocol.isoc.filter.<n>.offset

Bit offset for the ISO-C mask filter.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`32767`
Permissions	guest	–
	admin	`rw`

This variable specifies the bit offset for the ISO-C mask filter. It is the bit address where the tag will start the mask comparison.

modem.protocol.isoc.filter.<n>.session

Session target used for the ISO-C mask filter.

Class	`VAR`	
Type	`enum`	
	default value: `NOT_USED`	
Permissions	guest	`-`
	admin	`rw`

This variable is the session target for the ISO-C mask filter. This is the session inventoried flag or SL flag modified when a tag matches the filter. If set to `NOT_USED`, then the variables `modem.protocol.isoc.filtering.use_session` and `modem.protocol.isoc.control.session_id` will control whether a session or select flag is used for this filter. The priority of commands for a filter is as follows: If `modem.protocol.isoc.filter.x.session` is S0,S1,S2,S3 or SL then selected session is used in the filter. If `modem.protocol.isoc.filtering.use_session` is `true` AND `modem.protocol.isoc.filter.x.session` is `NOT_USED` then `modem.protocol.isoc.control.session_id` is used in the filter. If `modem.protocol.isoc.filtering.use_session` is `false` AND `modem.protocol.isoc.filter.x.session` is `NOT_USED` then the select flag (SL) is used in the filter. (Default configuration)

15.37 MODEM.PROTOCOL.ISOC.FILTERING

modem.protocol.isoc.filtering.enable

Enables the ISO-C mask filter.

Class	`VAR`	
Type	`bool`	
	default value: `False`	
Permissions	guest	`-`
	admin	`rw`

This variable enables the ISO-C mask filter. When set to `true`, only those ISO-C tags that match the filter mask will respond. The filter mask is a bit pattern which starts at a specific bit offset in the tag and has a specific length.

modem.protocol.isoc.filtering.truncated_epc_response

Turn on ISO-C truncation.

Class	`VAR`	
Type	`bool`	
	default value: `False`	
Permissions	guest	`-`
	admin	`rw`

This variable turns on ISO-C truncation. When truncation is `true` and filtering is enabled, the mask filter is used and tags that match the mask will truncate their EPC response to the bits that follow the mask. The mask must end in the EPC memory or truncation will not be enabled.

For truncation to work, the filter mask must start at offset 16 in EPC memory, which is the start of the Protocol-control (PC) bits. The most significant 5 bits of the PC indicate the length of the tags EPC in 16-bit words, plus the 1 word of PC bits. Only 1 filter mask can be used with truncation.

The following example sets up a filter and uses truncation to find 96-bit EPC tags with `tag_id=301020304050xxxxxxxxxxxx`, where x is don't care. The length is 64 bits where: 64 bits = 16 (PC bits) + 48 bits (EPC bits in mask). The offset is the offset to PC bits in EPC memory. The filter mask is 3000301020304050 where the first 16 bits are the PC bits from EPC memory.

```
modem.protocol.isoc.filter.1.length=64

modem.protocol.isoc.filter.1.offset=16

modem.protocol.isoc.filter.1.mask=30003010203
04050

modem.protocol.isoc.filter.1.action=assert_de
assert

modem.protocol.isoc.filter.1.enable=1

modem.protocol.isoc.filtering.truncated_epc_r
esponse=1

modem.protocol.isoc.filtering.enable=1
```

modem.protocol.isoc.filtering.truncated_tag_epc_length

This variable indicates the expected tag EPC length for truncation.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`512`
Permissions	guest	–
	admin	`rw`

This variable sets the expected tag EPC length for truncation. When set to 0, the EPC length is calculated from the PC word included in the filter mask. See `modem.protocol.isoc.filtering.truncated_epc_response.` Otherwise, this value is used.

modem.protocol.isoc.filtering.use_session

Target the tag session inventoried flag for ISO-C filter.

Class	`VAR`	
Type	`bool`	
	default value: `False`	
Permissions	guest	–
	admin	`rw`

This variable causes the ISO-C filter to target the tags session inventoried flag instead of the tags SL flag. Any tag action based on match/mismatch of the filter mask will cause the tag to change its inventoried flag for the session, and not the tags SL flag.

15.38 MODEM.PROTOCOL.ISOC.PHYSICAL

modem.protocol.isoc.physical.data_1_length

Length of a Data 1 symbol for ISO-C.

Class	VAR	
Type	enum	
	default value: D1_LEN_15	
Permissions	guest	r
	admin	r

This variable is the length of a Data 1 symbol for ISO-C. The ISO-C Data 1 symbol length can be set to either (Tari * 1.5) or (Tari * 2.0). To change, use the Set function.

modem.protocol.isoc.physical.modulation_depth

Modulation depth percentage for ISO-C.

Class	VAR	
Type	int	
	default value:	90
	min	0
	max	100
Permissions	guest	r
	admin	rw

This specifies the percent modulation depth for ISO-C transmissions. The ISO-C spec requires the depth to be between 80 % to 100%.

MODEM NAMESPACE

modem.protocol.isoc.physical.pilot_threshold

Pilot tone detect threshold in dBm

Class	`VAR`		
Type	`int`		
	default value:	`-60`	
	min	`-100`	
	max	`0`	
Permissions	guest	–	
	admin	`rw`	

Pilot tone detect threshold. If set to 0, the modem will automatically set the threshold based on the noise environment and signal bandwidth.

modem.protocol.isoc.physical.pilot_tone

Turn on ISO-C `pilot_tone` from tag.

Class	`VAR`	
Type	`bool`	
	default value: `True`	
Permissions	guest	`r`
	admin	`r`

This variable turns on the ISO-C `pilot_tone` from the tag. When `true`, the pilot tone is on and the tag will prepend 12 leading 0's to it's response preamble.

modem.protocol.isoc.physical.return_link_freq

Return link frequency from tag to reader.

Class	`VAR`	
Type	`enum`	
	default value: `LF240`	
Permissions	guest	`r`
	admin	`r`

This variable sets the return link frequency for the tag-to-reader communication. The actual data rate will also depend on the encoding (such as FM0, Miller-2, Miller-4, or Miller-8). To change, use the Set function.

modem.protocol.isoc.physical.rt_modulation

Reader to tag modulation type.

Class	`VAR`	
Type	`enum`	
	default value: `RT_MOD_PR`	
Permissions	guest	`r`
	admin	`r`

This variable is the reader-to-tag modulation type. The reader can transmit using either double sideband ASK or phase reversal ASK modulation.

modem.protocol.isoc.physical.set

Set ISOC physical link parameters.

Class	FUNCTION		
Parameters	tari	:	ENUM DEFINITIONS.ENUM.PROTOCOL.ISOC.PHYSICAL.TARI
	data_1_length	:	ENUM DEFINITIONS.ENUM.PROTOCOL.ISOC.PHYSICAL.DATA_1_LENGTH
	return_link_freq	:	ENUM DEFINITIONS.ENUM.PROTOCOL.ISOC.PHYSICAL.RETURN_LINK_FREQ
	tr_encoding	:	ENUM DEFINITIONS.ENUM.PROTOCOL.ISOC.PHYSICAL.TR_ENCODING
	rt_modulation	:	ENUM DEFINITIONS.ENUM.PROTOCOL.ISOC.PHYSICAL.RT_MODULATION
	modulation_depth	:	INT
	pilot_threshold	:	INT
	write_response_delay	:	INT
Response	BOOL		
Permissions	guest	–	
	admin	x	

This function sets ISOC physical link parameters. This function MUST be used when you wish to change the tari, `data_1_length`, or `return_link_freq` parameters. If any one of these parameters (tari, `data_1_length`, `return_link_freq`) are set, they must all be set.

modem.protocol.isoc.physical.settings

Holds ISO-C physical layer settings.

Class	`VAR`	
Type	`string`	
	default value: `""`	
Permissions	guest	`r`
	admin	`r`

This variable holds ISO-C physical layer settings so they can be saved and restored with one variable.

modem.protocol.isoc.physical.tari

ISO-C Tari value.

Class	`VAR`	
Type	`enum`	
	default value: `TARI_25_00`	
Permissions	guest	`r`
	admin	`r`

This variable specifies the Tari (in microseconds) used for the ISO-C physical layer. To change, use the Set function.

modem.protocol.isoc.physical.tr_encoding

Tag to reader encoding format.

Class	`VAR`	
Type	`enum`	
	default value: `TR_ENC_MILLER_4`	
Permissions	guest	`r`
	admin	`r`

This variable is the tag to reader encoding format. The reader instructs the tag to use 1 of 2 encoding formats: FM0 baseband or Miller sub-carrier.

modem.protocol.isoc.physical.write_response_delay

Maximum receive window delay time (milliseconds) after an ISOC write type command.

Class	VAR	
Type	int	
	default value:	20
	min	0
	max	20
Permissions	guest	r
	admin	rw

15.39 MODEM.PROTOCOL.PS111

modem.protocol.ps111.read

Read specified data from a PS111 tag.

Class	FUNCTION	
Parameters	**antenna** :	INT
Response	COMPOUND	
Permissions	guest	x
	admin	x

- antenna (optional) - Indicates antenna on which to execute the function. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name, followed by the data read from the tag.

modem.protocol.ps111.write

Write specified data to a PS111 tag.

Class	FUNCTION		
Parameters	**user_data**	:	ARRAY
	antenna	:	INT
Response	COMPOUND		
Permissions	guest	x	
	admin	x	

This function writes specified data to a PS111 tag in the field.

- `user_data` - if specified, this data will be written to the tag. Must contain full 256 bits of data, including read-only portion. If `modem.protocol.ps111.control.use_dynamic_write_data` is `true`, the dates/times/seq # of the write data will be overwritten prior to the write command.
- antenna (optional) - Indicates antenna on which to execute the function. If an antenna is not specified, all antennas will be tried until the function succeeds or no more antennas are available.

The response to this function is a response name.

15.40 MODEM.PROTOCOL.PS111.CONTROL

modem.protocol.ps111.control.frequency

Frequency to use for PS111 protocol.

Class	VAR	
Type	int	
	default value:	915750
	min	914250
	max	915750
Permissions	guest	-
	admin	rw

The reader will use the specified frequency (in kHz) when performing PS111 operations, then go back to the standard frequency scheme for other protocols. 915750 is the default frequency for this protocol.

modem.protocol.ps111.control.inventory_uses_retries

Enables or disables the retries mechanism for inventory.

Class	`VAR`	
Type	`bool`	
	default value: `FALSE`	
Permissions	guest	–
	admin	`rw`

When `true`, each PS111 inventory cycle will use retries number of attempts, or until tag is inventoried (read). When `false`, only one attempt at inventory is performed.

modem.protocol.ps111.control.new_tag_window

Defines how long a tag is considered to have already been read.

Class	`VAR`	
Type	`int`	
	default value: `60`	
Permissions	guest	–
	admin	`rw`

If a tag is read that has the same TC Agency ID, TC Plaza TD, and TC Lane ID, AND the TC Date and Time are within a certain window of the current date and time, the tag is considered to have already been read. This variable configures the time window, in seconds. (For example a value of 60 means that a tag is considered to have already been read if its TC Date and Time are less than 60 seconds old, and the Agency ID, Plaza ID, and Lane ID match the values of these parameters in the `write_data` namespace. See the `modem.protocol.ps111.control.report_occasion` and `modem.protocol.ps111.control.write_occasion` for variables that depend on the meaning of the new tag window value. See the `modem.protocol.ps111.control.write_data` namespace for the Agency ID and other parameters mentioned above.

modem.protocol.ps111.control.report_occasion

Controls when PS111 tag read reports occur.

Class	`VAR`	
Type	`enum`	
	default value: `ALWAYS`	
Permissions	guest	`–`
	admin	`rw`

This variable is used to control when PS111 tag read reports occur. When set to ALWAYS, the reader generates a read report every time a tag is read. When set to `WHEN_NEW`, the reader generates a read report only the first time a new tag is seen. See the `modem.protocol.ps111.control.new_tag_window` to define when a tag is considered to be new.

modem.protocol.ps111.control.retries

Sets the number of retries attempted for the PS111 protocol.

Class	`VAR`	
Type	`int`	
	default value: `3`	
Permissions	guest	`–`
	admin	`rw`

This variable controls how many attempts are made for a write, or write verification. A 0 indicates no retries (just try once), a 1 indicates one retry (for a total of two write attempts), etc. During normal inventory operation retries are not used to read the tag unless `modem.protocol.ps111.control.inventory_uses_retries` is set to `true`. If write verification is disabled, this variable has no effect on writes; writes are only attempted once. See the `modem.protocol.ps111.control.verify_write` variable to enable write verification.

modem.protocol.ps111.control.tx_atten

Transmit attenuation for PS111 in ddBm.

Class	VAR	
Type	int	
	default value:	0
	min	0
	max	300
Permissions	guest	r
	admin	rw

This specifies the transmit attenuation applied during PS111 transmissions. The value is in ddBm so a value of 200 would be an attenuation of 20 dBm.

modem.protocol.ps111.control.use_dynamic_write_data

Enables or disables using dynamic time/date/etc

in PS111 writes.

Class	VAR	
Type	bool	
	default value: TRUE	
Permissions	guest	–
	admin	rw

When `false`, the TM Date, TM Time, TC Date, TC Time, and TC Seq Num values written to the tag are the values defined in the corresponding `modem.protocol.ps111.control.write_data` variable. When `true`, these fields are written with values generated on-the-fly by the reader. The TC Lane ID value is always generated from the list of `lane_id` values defined by `modem.protocol.ps111.control.write_data.tc_lane_id` except when the `use_dynamic_write_data` is `false` AND the user is calling the write() function with explicit user data, in which case the user data value is used.

modem.protocol.ps111.control.verify_write

Enables or disabled PS111 write verification.

Class	`VAR`	
Type	`bool`	
	default value: `TRUE`	
Permissions	guest	`–`
	admin	`rw`

When `true`, each PS111 write is followed by a read to verify the data was written correctly. When `false`, no write verification is performed.

modem.protocol.ps111.control.write_occasion

Controls when PS111 writes occur.

Class	`VAR`	
Type	`enum`	
	default value: `WHEN_NEW`	
Permissions	guest	`–`
	admin	`rw`

This variable is used to control when PS111 writes occur. When set to NEVER, the reader never automatically performs a write - the write() function is the only way to cause a write. When set to ALWAYS, the reader performs a write to the tag every time a tag is read. When set to `WHEN_NEW`, the reader performs a write only the first time a new tag is seen. See the `modem.protocol.ps111.control.write_data` namespace to control what data is written to the tag. See the `modem.protocol.ps111.control.new_tag_window` to define when a tag is considered to be new.

modem.protocol.ps111.control.write_type

Controls which portion of PS111 tag memory will be written.

Class	`VAR`	
Type	`enum`	
	default value: `TRAFFIC_AND_TOLL`	
Permissions	guest	–
	admin	`rw`

This variable indicates whether the reader will write to the Traffic Management ™ section of R/W tag memory, the Toll Collection (TC) section, or both, whenever the tag is written.

the protocol dictates the entire tag must be written each time, but the reader will simply echo back the read data for the areas of memory which are not being actively written to.

15.41 MODEM.PROTOCOL.PS111.CONTROL.WRITE_DATA

modem.protocol.ps111.control.write_data.tc_agency_data

Agency Data to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0x3fffffff`
Permissions	guest	–
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Agency Data field.

modem.protocol.ps111.control.write_data.tc_agency_id

Agency ID to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0x7f`
Permissions	guest	–
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Agency ID field.

modem.protocol.ps111.control.write_data.tc_date

Date to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0x1ff`
Permissions	guest	–
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Date field, if `modem.protocol.ps111.control.use_dynamic_data` is `false`.

modem.protocol.ps111.control.write_data.tc_future

Future Data to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0xf`
Permissions	guest	–
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Future Data field.

modem.protocol.ps111.control.write_data.tc_lane_id

Lane ID(s) to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`list`	
	default value: `0`	
Permissions	guest	–
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Lane ID field.

the variable is a list; each element in the list indicates the Lane ID used for the corresponding antenna. The TC Lane ID value is always generated from the list of `lane_id` values defined by `modem.protocol.ps111.control.write_data.tc_lane_id` except when the `use_dynamic_write_data` is `false` AND the user is calling the write() function with explicit user data, in which case the user data value is used.

modem.protocol.ps111.control.write_data.tc_plaza_id

Plaza ID to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0x7f`
Permissions	guest	–
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Plaza ID field.

modem.protocol.ps111.control.write_data.tc_seq_num

Seq / Txn Number to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0xffff`
Permissions	guest	–
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Seq / Txn Number field, if `modem.protocol.ps111.control.use_dynamic_data` is `false`.

modem.protocol.ps111.control.write_data.tc_time

Time to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0x1ffff`
Permissions	guest	–
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Time field, if `modem.protocol.ps111.control.use_dynamic_data` is `false`.

modem.protocol.ps111.control.write_data.tc_vehicle_class

Vehicle Class to be written into the Toll Collection (TC) tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0x7ff`
Permissions	guest	–
	admin	`rw`

If and when Toll Collection data is written to a PS111 tag, the value of this variable will be used to write the Vehicle Class field.

modem.protocol.ps111.control.write_data.tm_date

Date to be written into the Traffic Management ™ tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0x1ff`
Permissions	guest	–
	admin	`rw`

If and when Traffic Management data is written to a PS111 tag, the value of this variable will be used to write the Date field if `modem.protocol.ps111.control.use_dynamic_write_data` is `false`.

modem.protocol.ps111.control.write_data.tm_reader_id

Reader ID to be written into the Traffic Management ™ tag memory.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`0xfff`
Permissions	guest	–
	admin	`rw`

If and when Traffic Management data is written to a PS111 tag, the value of this variable will be used to write the Reader ID field.

modem.protocol.ps111.control.write_data.tm_time

Time to be written into the Traffic Management ™ tag memory.

Class	`VAR`		
Type	`int`		
		default value:	`0`
		min	`0`
		max	`0x1ffff`
Permissions	guest	–	
	admin	`rw`	

If and when Traffic Management data is written to a PS111 tag, the value of this variable will be used to write the Time field if `modem.protocol.ps111.control.use_dynamic_write_data` is `false`.

15.42 MODEM.PROTOCOL.PS111.PHYSICAL

modem.protocol.ps111.physical.error_threshold

Max decode error threshold for PS111.

Class	`VAR`	
Type	`int`	
	default value: `0`	
Permissions	guest	–
	admin	`rw`

Abort demodulation when cumulative timing error exceeds this value. (0 = disabled; do not abort demodulation). 10000000 might be a good starting point for this variable, if used.

modem.protocol.ps111.physical.fixed_lna

Set fixed value for LNA in PS111 mode.

Class	VAR	
Type	int	
	default value:	0
	min	-1
	max	1
Permissions	guest	–
	admin	rw

Force LNA on/off in PS111 mode. When 0, LNA forced off for PS111. When 1, LNA forced on. When -1, LNA set based on AGC.

modem.protocol.ps111.physical.fixed_vga_dac

Set fixed value for VGA DAC in PS111 mode.

Class	VAR	
Type	int	
	default value:	1
	min	-1
	max	2
Permissions	guest	–
	admin	rw

Force VGA to low/med/high gain in PS111 mode. When 0, VGA DAC forced to 0. When 1, VGA DAC forced to 512. When 2, VGA DAC forced to 1023. When -1, VGA DAC based on AGC.

modem.protocol.ps111.physical.pilot_threshold

Detection threshold for PS111, in dBm

Class	`VAR`	
Type	`int`	
	default value:	`-60`
	min	`-100`
	max	`0`
Permissions	guest	`-`
	admin	`rw`

Detection threshold (sensitivity) for PS111 tags. Increase to a higher value to reduce sensitivity, which should also reduce CRC errors. A value of 0 indicates the standard/default threshold should be used.

15.43 MODEM.PROTOCOL.T21.CONTROL

modem.protocol.t21.control.ack_tag

Acknowledge Title 21 tag.

Class	`VAR`	
Type	`bool`	
	default value: `True`	
Permissions	guest	`r`
	admin	`rw`

This variable enables acknowledgement of a Title 21 tag. When set to `true`, an Ack command is sent to the tag to quiet it for a set duration. When set to `false`, no Ack is sent and the tag can continue responding to Poll commands.

modem.protocol.t21.control.ack_with_noop

Acknowledge Title 21 tag with a NOOP record.

Class	`VAR`	
Type	`bool`	
	default value: `False`	
Permissions	guest	`r`
	admin	`rw`

This variable enables acknowledgement of a Title 21 tag using a NOOP transaction record type code (`0xC001`) instead of the standard `0xC000` record type. When set to `true`, an Ack command is sent to the tag with the NOOP record and the tag is not quieted. When set to `false`, the standard record type is sent in the acknowledge message to quiet the tag. The variable `modem.protocol.t21.control.ack_tag` has precedence over `ack_with_noop`, so if `ack_tag` is `false`, no Acks are sent regardless of `ack_with_noop` setting.

modem.protocol.t21.control.display_tag_crc

Display of tag CRC.

Class	`VAR`	
Type	`bool`	
	default value: `False`	
Permissions	guest	`r`
	admin	`rw`

This variable enables display of the tag CRC. When set to `true`, entire tag data contents, including record type and CRC, is displayed. When set to `false`, only the tag ID bits are displayed.

MODEM NAMESPACE

modem.protocol.t21.control.enable_crc_check

Enable CRC checking of tag response.

Class	VAR	
Type	bool	
	default value: True	
Permissions	guest	r
	admin	rw

This variable enables CRC checking of the tag response. When set to `true`, the received data is verified with the received CRC. When set to `false`, the CRC checking is disabled.

modem.protocol.t21.control.encrypt_ack_data

Encrypt mode for data sent in ACK message.

Class	VAR	
Type	enum	
	default value: LEGACY	
Permissions	guest	r
	admin	rw

This variable enables encryption of the data sent in the ACK message for Title 21 tags. When set to legacy, no encryption is used and the `reader_id` and `status_code` of the ACK message are based on their variable settings. When set to clear, support for switchable SOV mode tags is enabled with the data in the ACK sent unencrypted. When set to encrypted, the data is sent encrypted.

modem.protocol.t21.control.extra_inventory_cycles

Number of extra inventory cycles per round for Title 21

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`50`
Permissions	guest	`r`
	admin	`rw`

This specifies the number of extra inventory cycles per round for Title 21 protocol. When 0 (default), there is one inventory cycle performed. Each increment of this variable will cause an additional Title 21 inventory cycle to be performed during a single round.

modem.protocol.t21.control.status_code

Transponder status code.

Class	`VAR`	
Type	`string`	
	default value:	`"0x0000"`
Permissions	guest	`r`
	admin	`rw`

This variable contains the status code data sent to the tag in Title 21 Acknowledge messages.

modem.protocol.t21.control.tx_atten

Transmit attenuation for Title 21 in ddBm.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`300`
Permissions	guest	`r`
	admin	`rw`

This specifies the transmit attenuation applied during Title 21 transmissions. The value is in ddBm so a value of 200 would be an attenuation of 20 dBm.

15.44 MODEM.PROTOCOL.T21.PHYSICAL

modem.protocol.t21.physical.fixed_lna

Set fixed value for LNA in Title-21 mode.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`-1`
	max	`1`
Permissions	guest	`-`
	admin	`rw`

Force LNA on/off in Title-21 mode. When 0, LNA forced off for Title-21. When 1, LNA forced on. When -1, LNA set based on AGC.

modem.protocol.t21.physical.fixed_vga_dac

Set fixed value for VGA DAC in Title-21 mode.

Class	VAR	
Type	int	
	default value:	1
	min	-1
	max	2
Permissions	guest	–
	admin	rw

Force VGA to low/med/high gain in Title-21 mode. When 0, VGA DAC forced to 0. When 1, VGA DAC forced to 512. When 2, VGA DAC forced to 1023. When -1, VGA DAC based on AGC.

modem.protocol.t21.physical.modulation_depth

Modulation depth percentage for Title-21.

Class	VAR	
Type	int	
	default value:	100
	min	0
	max	100
Permissions	guest	r
	admin	rw

This variable specifies the percent modulation depth for Title-21 transmissions.

modem.protocol.t21.physical.pilot_threshold

Pilot tone detect threshold in dBm

Class	`VAR`	
Type	`int`	
	default value:	`-70`
	min	`-100`
	max	`0`
Permissions	guest	–
	admin	`rw`

Pilot tone detect threshold. If set to 0 (default), the modem will automatically set the threshold based on the noise environment and signal bandwidth.

15.45 MODEM.RADIO

modem.radio.idle_cw

Transmit CW during idle time.

Class	`VAR`	
Type	`bool`	
	default value: `False`	
Permissions	guest	–
	admin	`rw`

When the read is idle, not talking to tags, the reader can either transmit CW or can turn RF off. If `idle_cw` is set to `True`, the reader will trasmit CW when it is idle.

modem.radio.rf_survey

Survey RF environment.

Class	FUNCTION		
Parameters	**rbw**	:	ENUM DEFINITIONS.ENUM.RESOLUTION_BW
	span	:	ENUM DEFINITIONS.ENUM.FREQ_SPAN
	coherent	:	BOOL
	center_freq	:	INT
	output_freq	:	INT
Response	COMPOUNDLIST		
Permissions	guest	–	
	admin	x	

This function performs a spectrum analysis at the current frequency. The results of this function are scaled to account for all system gains. The parameters supported by this function are: * rbw - Resolution bandwidth

- span - Frequency span
- coherent - Enable coherent scaling
- `center_freq` - Specify the center frequency. If not specified the current frequeny is used.
- `output_freq` - Specify the output frequency. If not specified all frequencies will be returned.

15.46 MODEM.RADIO.FREQ_MGMT.HOP_TABLE

modem.radio.freq_mgmt.hop_table.frequencies

Frequency hop table

Class	`VAR`	
Type	`list`	
	default value: `0`	
Permissions	guest	–
	admin	`r`

This is a list of the frequencies (in kHz) used in the current hop table. Either this variable OR the .start, .stop, .step, .mask variables should be used.

15.47 MODEM.RADIO.TX

modem.radio.tx.interlock

Regulatory interlock.

Class	`VAR`	
Type	`bool`	
	default value: `FALSE`	
Permissions	guest	–
	admin	`r`

The regulatory interlock is set if some condition on the board has put the reader at risk of violating RF regulatory standards. RcRfLowLevelOn and Off can be called by higher level functions without having violating regulatory requirements.

If a regulatory violation is detected, the interlock will not let RF be turned on. If activating an interlock, this variable will not return until RF ramping has completed.

modem.radio.tx.min_off_time

Minimum off time from TX ramp off to TX ramp on

Class	`VAR`	
Type	`int`	
	default value: `0`	
Permissions	guest	`–`
	admin	`rw`

This variable sets the minimum off time for the transmitter. Certain protocols require the transmitter to remain off for a specified period of time.

15.48 MODEM.STATS

modem.stats.tag_block_erase

This variable reports the total number of tags having the block erased by the modem.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`INT_MAX`
Permissions	guest	`r`
	admin	`rw`

modem.stats.tag_block_erase_failure

This variable reports the total number of tags failed the block erase.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`INT_MAX`
Permissions	guest	`r`
	admin	`rw`

modem.stats.tag_block_permalock

This variable reports the total number of tags having the block permalocked by the modem.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`INT_MAX`
Permissions	guest	`r`
	admin	`rw`

modem.stats.tag_block_permalock_failure

This variable reports the total number of tags failed the block permalock.

Class	VAR	
Type	int	
	default value:	0
	min	0
	max	INT_MAX
Permissions	guest	r
	admin	rw

modem.stats.tag_block_write

This variable reports the total number of tags having the block written by the modem.

Class	VAR	
Type	int	
	default value:	0
	min	0
	max	INT_MAX
Permissions	guest	r
	admin	rw

modem.stats.tag_block_write_failure

This variable reports the total number of tags failed the block write.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`INT_MAX`
Permissions	guest	`r`
	admin	`rw`

modem.stats.tag_kill

This variable reports the total number of tags killed by the modem.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`INT_MAX`
Permissions	guest	`r`
	admin	`rw`

modem.stats.tag_kill_failure

This variable reports the total number of tag kill failure.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`INT_MAX`
Permissions	guest	`r`
	admin	`rw`

modem.stats.tag_lock

This variable reports the total number of tags locked by the modem.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`INT_MAX`
Permissions	guest	`r`
	admin	`rw`

modem.stats.tag_lock_failure

This variable reports the total number of tag lock failure.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`INT_MAX`
Permissions	guest	`r`
	admin	`rw`

This variable reports the total number of tag locke failure.

modem.stats.tag_read

This variable reports the total number of tags read by the modem

The total counts multiple reads of the same tag as unique tag reads.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`INT_MAX`
Permissions	guest	`r`
	admin	`rw`

modem.stats.tag_read_failure

This variable reports the total number of tag read failure.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`INT_MAX`
Permissions	guest	`r`
	admin	`rw`

modem.stats.tag_singulation

This variable reports the total number of tag singulations read by the modem.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`INT_MAX`
Permissions	guest	`r`
	admin	`rw`

modem.stats.tag_singulation_failure

This variable reports the total number of tag singulation failure.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`INT_MAX`
Permissions	guest	`r`
	admin	`rw`

modem.stats.tag_write

This variable reports the total number of tags written by the modem.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`INT_MAX`
Permissions	guest	`r`
	admin	`rw`

modem.stats.tag_write_failure

This variable reports the total number of tag write failure.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`INT_MAX`
Permissions	guest	`r`
	admin	`rw`

modem.stats.uptime

This variable reports the number of seconds the modem has been running.

Class	`VAR`	
Type	`int`	
	default value:	`0`
	min	`0`
	max	`INT_MAX`
Permissions	guest	`r`
	admin	`r`

This variable reports the elapsed time (in seconds) the modem has been running. It is reset to 0 at startup and is updated once per second.

15.49 MODEM.VERSION

16 USER NAMESPACE

16 USER NAMESPACE

16.1 USER

user.var1

A variable for user access.

Class	`VAR`	
Type	`string`	
	default value: `""`	
Permissions	guest	`r`
	admin	`rw`

This variable is for user access. It can be used to hold anything a users wants.

17 ERROR NAMESPACE

17 ERROR NAMESPACE

The error namespace is a namespace tree. Each nested namespace provides more information about the specific error. For example the errors in the `error.file` namespace indicate there was an error processing a file. Errors in the `error.parser` namespace indicate there was an error parsing the command due to a mistake with the command's syntax.

The nested namespaces within the error namespace are:

- app
- file
- flash
- fwUpdate
- internal
- mode
- modem
- network
- parser

17.1 ERROR.APP

error.app.delete_failure

Attempt to delete an app failed.

Type `RESPONSE`

This response indicates an attempt to delete an app on the reader failed.

error.app.list_failure

Attempt to list running apps failed.

Type `RESPONSE`

This response indicates an attempt to list running apps on the reader failed.

error.app.not_running

Attempt to stop an application that is not running.

Type RESPONSE

This response indicates an attempt to stop a user application failed because the application is not running.

error.app.start_failure

Attempt to start an application failed.

Type RESPONSE

This response indicates an attempt to start a user application failed.

error.app.stop_failure

Attempt to stop an application failed.

Type RESPONSE

This response indicates an attempt to stop a user application failed.

17.2 ERROR.FILE

error.file.close_failure

Closing a file on the reader failed.

Type RESPONSE

This response indicates a command failed to close the file.

error.file.delete_failure

Deleting a file from the reader failed.

Type RESPONSE

This response indicates a command failed to delete a file.

error.file.link_failure

Linking a file on the reader failed.

Type `RESPONSE`

This response indicates a command failed to link a file.

error.file.open_failure

Opening a file on the reader failed.

Type `RESPONSE`

This response indicates a command failed to open a file.

error.file.stat_failure

Stat on a file on the reader failed.

Type `RESPONSE`

This response indicates a command failed to stat a file.

error.file.too_many_profiles

The maximum number of profiles has been exceeded.

Type `RESPONSE`

This response indicates a command to save a profile failed because the maximum number of profiles has been exceeded.

17.3 ERROR.FLASH

error.flash.read_failure

Operation failed because reader could not read flash memory.

Type `RESPONSE`

This response indicates the operation failed because the reader could not read flash memory.

error.flash.write_failure

Operation failed because reader could not write flash memory.

Type `RESPONSE`

This response indicates the operation failed because the reader could not write flash memory.

17.4 ERROR.FWUPDATE

error.fwupdate.processing_error

Error occured durring firmware change.

Type `RESPONSE`

This response indicates an error occured during a firmware change such as an upgrade or a rollback. More information about the error is supplied with the response.

17.5 ERROR.INTERNAL

error.internal.out_of_memory

Operation failed due to lack out of memory.

Type `RESPONSE`

This response indicates the operation failed because the reader ran out of memory needed for the operation.

error.internal.processing_error

Operation failed due to an internal processing error.

Type `RESPONSE`

This response indicates the operation failed because of an internal processing error.

17.6 ERROR.LICENSE

error.license.not_found

License necessary to perform operation has not been found, please contact Neology to obtain a license

Type `RESPONSE`

17.7 ERROR.MODE

error.mode.command_in_invalid_mode

Command rejected because incorrect reader mode.

Type `RESPONSE`

This response indicates the command was rejected because the reader was in the wrong mode.

17.8 ERROR.MODEM

error.modem.calibration_failed

The calibration has failed

A required device failed to respond or a value was out of range indicating a part failure.

Type `RESPONSE`

error.modem.invalid_frequency_table

Invalid frequency hop table specified.

Type `RESPONSE`

This response indicates an invalid frequency hop table was specified. Typically, this is because the start, stop, and step parameters were incompatible or because too many frequencies were specified. The info parameter will provide more information.

error.modem.invalid_response

Invalid modem response.

Type `RESPONSE`

This response indicates an invalid response was received from the modem.

error.modem.ppi_failure

Modem ppi control problem.

Type `RESPONSE`

This response indicates a command was sent to the modem and the modem had a ppi control problem.

error.modem.radio_control_failure

Modemradio control problem.

Type `RESPONSE`

This response indicates a command was sent to the modem and the modem had a radio control problem.

error.modem.rf_memory_failure

The memory on the RF board has failed.

Type `RESPONSE`

The RF board memory has failed. This memory holds the calibration data for the reader and diminished reader performance should be expected.

error.modem.spi_failure

Modem spi control problem.

Type `RESPONSE`

This response indicates a command was sent to the modem and the modem had a spi control problem.

error.modem.timeout

No modem response.

Type RESPONSE

This response indicates a command was sent to the modem and the modem did not respond within the expected time.

17.9 ERROR.NETWORK

error.network.address_not_reachable

Reader could not ping a network address.

Type RESPONSE

This response indicates the reader could not ping a network address.

error.network.failed_to_configure

Command to configure the network settings failed.

Type RESPONSE

This response indicates a command to configure network settings failed.

17.10 ERROR.PARSER

error.parser.antenna_power_negative

Antenna power is negative.

Type RESPONSE

This response indicates antenna power settings result in an negative power value.

17 ERROR NAMESPACE

error.parser.command_too_long

Command too long to be processed.

Type `RESPONSE`

This response indicates a command was too long (too many characters) to be processed.

error.parser.illegal_parameter

Set of supplied parameters is illegal.

Type `RESPONSE`

This response indicates the set of supplied parameters is illegal.

error.parser.illegal_parameter_type

Command uses an invalid parameter type.

Type `RESPONSE`

This response indicates the command uses an invalid parameter type.

error.parser.illegal_value

Illegal value in command.

Type `RESPONSE`

This response indicates one of the values in the command was illegal. For example, the command was expecting an INT and got a LIST instead.

error.parser.malformed_command

Command not presented in correct format.

Type `RESPONSE`

This response indicates a command was not in the correct format.

17 ERROR NAMESPACE

error.parser.missing_parameter

Command parameter missing.

Type `RESPONSE`

This response indicates the command was missing a parameter.

error.parser.permission_failure

You don't sufficient permissions for this operation.

Type `RESPONSE`

You have attempted a command for which you don't have sufficient permissions. Log in at a higher level to perform this action.

error.parser.processing_error

Error encountered during `processing..`

Type `RESPONSE`

This response indicates an error was encountered during processing. It is the default error in cases where there is no tranlsation for an internal error into a response code.

error.parser.response_too_long

Not enough memory to return full reponse.

Type `RESPONSE`

This response indicates the response was too long and could not be returned.

error.parser.set_failure

The `namespace.set()` function failed.

Type `RESPONSE`

This response indicates a `namespace.set()` function failed to set the specified value for at least one of the supplied parameters.

error.parser.unknown_command

Command was not recognized.

Type `RESPONSE`

This response indicates the command was not recognized.

error.parser.unknown_parameter

Command contained an unknown parameter.

Type `RESPONSE`

This response indicates the command contained an unknown parameter.

error.parser.unknown_variable

Command contained an unknown variable.

Type `RESPONSE`

This response indicates the command contained an unknown variable.

error.parser.value_out_of_range

Command value out of range.

Type `RESPONSE`

This response indicates a command value was out of range.

17.11 ERROR.RESOURCE

error.resource.busy

An external resource is busy.

Type `RESPONSE`

An external resource is temporarily unavailable. Try again later.

17.12 ERROR.SERIAL_PORT

error.serial_port.failed_to_configure

Command to configure the serial port failed.

Type `RESPONSE`

This response indicates a command to configure the serial port failed.

17.13 ERROR.TAG

error.tag.access_denied

Tag operation not allowed because of tag locks.

Type `RESPONSE`

This response indicates an operation was not allowed because of tag locks.

error.tag.buffer_overflow

The tag response overflows the receive buffer.

Type `RESPONSE`

This response indicates a tag operation failed because the tag response to a command is too large for the receive buffer.

error.tag.loss_of_signal

Tag operation failed because tag signal was lost.

Type `RESPONSE`

This response indicates a tag operation failed because the tag signal was lost.

17 ERROR NAMESPACE

error.tag.no_tag_in_field

Tag operation failed because no tag was found.

Type `RESPONSE`

This response indicates a tag operation failed because no tag was found.

error.tag.not_responding

Tag operation failed because tag not responding.

Type `RESPONSE`

This response indicates a tag operation failed because the tag has stopped responding.

error.tag.security_operation_failed

Tag security operation failed.

Type `RESPONSE`

This response indicates a tag security operation failed because the tag has not been commissioned or tag read/write/lock/… operation could not be completed.

error.tag.tag_not_writable

Tag operation failed because tag not writable.

Type `RESPONSE`

This response indicates a tag operation failed because the tag was not writable.

error.tag.unsupported_command

Tag operation failed because the command is not supported by the tag.

Type `RESPONSE`

This response indicates a tag operation failed because the command is not supported by the tag.

error.tag.verify_failed

Tag verify operation failed.

Type `RESPONSE`

This response indicates a tag verify operation (read back after write) failed.

17.14 ERROR.TAG.PROTOCOL.ISOC

error.tag.protocol.isoc.insufficient_power

ISO-C tag operation failed because of insufficient power.

Type `RESPONSE`

This response indicates an ISO-C tag operation failed because the tag has insufficient power to perform the memory-write operation.

error.tag.protocol.isoc.memory_locked

ISO-C tag operation failed because memory is locked.

Type `RESPONSE`

This response indicates an ISO-C tag operation failed because the specified memory location is locked or permalocked and not writeable or readable.

error.tag.protocol.isoc.memory_overrun

ISO-C tag operation failed due to no memory.

Type `RESPONSE`

This response indicates an ISO-C tag operation failed because the specified memory location does not exist or the PC value is not supported by the tag.

error.tag.protocol.isoc.non_specific

ISO-C tag does not support error-specific codes.

Type `RESPONSE`

This response indicates an ISO-C tag does not support error-specific codes.

error.tag.protocol.isoc.other_error

ISO-C tag operation failed.

Type `RESPONSE`

This response indicates an ISO-C tag operation failed with no specific error

18 EVENTS NAMESPACE

18 EVENTS NAMESPACE

18.1 EVENT

event.connection

New event channel established with reader.

Class	`EVENT`		
Types	**id**	:	`INT`
Permissions	guest	`rw`	
	admin	`rw`	

This event is generated when a new event channel has been established with the reader. The new event channel is identified by the ID parameter passed back in this event. The ID can be used to register for events on this event channel with the bind and registerevent commands.

event.info

Informational event.

Class	`EVENT`		
Types	**id**	:	`INT`
	time	:	`STRING`
	text	:	`STRING`
Permissions	guest	`rw`	
	admin	`rw`	

These events go into the error/warning log but are informational only, and do not indicate an error or warning condition.

18.2 EVENT.CONFIGURATION

event.configuration.change

Configuration change event.

Class	EVENT		
Types	**name**	:	STRING
	newvalue	:	STRING
	oldvalue	:	STRING
Permissions	guest	rw	
	admin	rw	

This event is generated each time a configuration is changed.

18.3 EVENT.DEBUG.TRACE

event.debug.trace.iop

Debug tracing information sent from reader.

Class	EVENT		
Types	**file**	:	STRING
	line	:	INT
	time	:	STRING
	data	:	STRING
Permissions	guest	rw	
	admin	rw	

This event sends debug tracing information from the reader. The following example illustrates an event.debug.trace.iop event. event.debug.trace.iop file=TagDatabase.c line=203 time=01/03/06-08:56:13 data='tag hashed to value 482'

event.debug.trace.modem

This event sends debug tracing information from the modem out of the reader.

Class	`EVENT`		
Types	**file**	:	`STRING`
	line	:	`INT`
	modem_ms	:	`INT`
	data	:	`STRING`
Permissions	guest	`rw`	
	admin	`rw`	

This event sends debug tracing information from the modem. The `modem_ms` is the number of ms the modem has run since bootup (rough timestamp from modem DSP). An example of an event.debug.trace.modem event would look like: event.debug.trace.modem file=mem.c line=804 modem_ms=827443 data='Free'ed a block which was not previously allocated!'

18.4 EVENT.DIO

event.dio.all

Report the state of all digital inputs and outputs.

Class	`EVENT`		
Types	**in**	:	`INT`
	out	:	`INT`
Permissions	guest	`rw`	
	admin	`rw`	

This event reports the state of all digital IOs and is generated each time a digital IO changes.

18.5 EVENT.DIO.IN

event.dio.in.<n>

Digital IO event.

Class	`EVENT`		
Types	**value**	:	`INT`
Permissions	guest	`rw`	
	admin	`rw`	

This event is generated each time a digital input changes.

18.6 EVENT.DIO.IN.ALARM.TIMEOUT

event.dio.in.alarm.timeout.<n>

Digital IO event.

Class	`EVENT`		
Types	**value**	:	`INT`
	logic_level	:	`INT`
	timeout	:	`INT`
Permissions	guest	`rw`	
	admin	`rw`	

If a timeout value is set, the input pin is monitored. If the input pin value does not change during the timeout period AND the input pin value matches the alarm logic level, the event `event.dio.in.alarm.timeout.n` (where n is the pin number) is generated. This alarm event generation can be helpful in alerting to the loss of digital inputs to the reader.

18.7 EVENT.DIO.OUT

event.dio.out.<n>

Digital IO event.

Class	`EVENT`	
Types	**value** : `INT`	
Permissions	guest	`rw`
	admin	`rw`

This event is generated each time a digital output changes.

18.8 EVENT.ERROR

event.error.antenna

Reader detected an error condition with the antennas.

Class	`EVENT`	
Types	**id** : `INT`	
	time : `STRING`	
	text : `STRING`	
Permissions	guest	`rw`
	admin	`rw`

This event indicates that the reader detected an error condition with the antennas.

18 EVENTS NAMESPACE

event.error.communication

Reader detected a communication error.

Class	EVENT		
Types	id	:	INT
	time	:	STRING
	text	:	STRING
Permissions	guest	rw	
	admin	rw	

This event indicates that the reader detected a communication error.

event.error.configuration

Reader detected a configuration error.

Class	EVENT		
Types	id	:	INT
	time	:	STRING
	text	:	STRING
Permissions	guest	rw	
	admin	rw	

This event indicates that the reader detected a configuration error.

event.error.environmental

Reader detected an error condition with the environmental condition around the reader.

Class	EVENT		
Types	id	:	INT
	time	:	STRING
	text	:	STRING
Permissions	guest	rw	
	admin	rw	

This event indicates that the reader detected an error condition with the environmental condition around the reader.

event.error.file_handling

Reader detected an error condition while operating on a file.

Class	`EVENT`		
Types	**id**	:	`INT`
	time	:	`STRING`
	text	:	`STRING`
Permissions	guest	`rw`	
	admin	`rw`	

This event indicates that the reader detected an error condition while operating on a file.

event.error.hw

Reader detected an error condition with the hardware.

Class	`EVENT`		
Types	**id**	:	`INT`
	time	:	`STRING`
	text	:	`STRING`
Permissions	guest	`rw`	
	admin	`rw`	

This event indicates that the reader detected an error condition with the hardware.

18 EVENTS NAMESPACE

event.error.radio

Reader detected an error condition with the radio.

Class	EVENT		
Types	**id**	:	INT
	time	:	STRING
	text	:	STRING
Permissions	guest	rw	
	admin	rw	

This event indicates that the reader detected an error condition with the radio.

event.error.sw

Reader detected an error condition with the software.

Class	EVENT		
Types	**id**	:	INT
	time	:	STRING
	text	:	STRING
Permissions	guest	rw	
	admin	rw	

This event indicates that the reader detected an error condition with the software.

18.9 EVENT.PROFILE

event.profile.change

Profile change event.

Class	EVENT		
Types	**newprofile**	:	STRING
	oldprofile	:	STRING
Permissions	guest	rw	
	admin	rw	

This event is generated each time a new profile is loaded or resettig to factory profile.

18.10 EVENT.RESPONSE

event.response.modem_dio_scripts

Modem response for each dio script command.

Class	EVENT		
Types	**dio_in**	:	INT
	cmdnum	:	INT
	resp	:	STRING
Permissions	guest	rw	
	admin	rw	

Modem response from the command defined in `modem.dio.in.*.script.\*.`

18.11 EVENT.STATUS

event.status.antenna_return_loss

Antenna return loss.

Class	EVENT		
Types	**antenna**	:	INT
	return_loss	:	INT
Permissions	guest	rw	
	admin	rw	

This event indicates the return loss of an antenna. The antenna port, return loss are provided.

event.status.antenna_state_change

Antenna state change.

Class	EVENT		
Types	**antenna**	:	INT
	previous state	:	STRING
	current state	:	STRING
Permissions	guest	rw	
	admin	rw	

This event indicates an antenna has transitioned to a new state. The antenna port, previous state, and current state are provided.

18 EVENTS NAMESPACE

event.status.antenna_transition

Reader has transitioned to new antenna port.

Class	`EVENT`		
Types	**prev_antenna**	:	`INT`
	new_antenna	:	`INT`
	time	:	`STRING`
Permissions	guest	`rw`	
	admin	`rw`	

This event indicates the reader has transitioned to a new antenna port. The new antenna port is identified by the `new_antenna` parameter. The previous antenna port is also provided, as well as the time (in ms) the previous antenna port was used prior to the transition.

event.status.channel_scan

Report power spectrum estimate on current channel/antenna

Class	`EVENT`		
Types	**antenna**	:	`INT`
	frequency	:	`INT`
	p	:	`INT`
	gain	:	`INT`
Permissions	guest	`r`	
	admin	`rw`	

event.status.channel_state

Report a change in state for a channel.

Class	EVENT		
Types	**new_frequency**	:	INT
	old_state	:	ENUM
	new_state	:	ENUM
	t	:	INT
Permissions	guest	r	
	admin	rw	

This event indicates a channel changed state.

event.status.channel_transition

Reader has transitioned to new channel.

Class	EVENT		
Types	**prev_frequency**	:	INT
	new_frequency	:	INT
	time	:	INT
Permissions	guest	rw	
	admin	rw	

This event indicates the reader has transitioned to a new channel. The channel is identified by the `new_frequency` parameter. The previous channel is also provided, as well as the time (in ms) the previous channel used prior to the transition.

event.status.inventory_end

Reader has completed an inventory round.

Class	`EVENT`		
Types	**tag_type**	:	`STRING`
	total_slots	:	`INT`
	empty_slots	:	`INT`
Permissions	guest	`rw`	
	admin	`rw`	

This event indicates the reader has completed an inventory round.

event.status.inventory_rounds_complete

Inventory rounds have completed.

Class	`EVENT`	
Types		
Permissions	guest	`rw`
	admin	`rw`

The inventory rounds started with a `modem.control.inventory.perform_rounds` function have completed.

event.status.inventory_start

Reader has started an inventory round.

Class	`EVENT`		
Types	**tag_type**	:	`STRING`
Permissions	guest	`rw`	
	admin	`rw`	

This event indicates the reader has started an inventory round.

18 EVENTS NAMESPACE

event.status.modem_halted

The modem of the reader detected an unrecoverable error and halted.

Class	`EVENT`	
Types		
Permissions	guest	`rw`
	admin	`rw`

This event indicates the modem has detected an unrecoverable error and halted. No further communication with the modem is possible after this event.

event.status.modem_ready

The modem of the reader is ready to process commands.

Class	`EVENT`	
Types		
Permissions	guest	`rw`
	admin	`rw`

This event indicates the modem has finished initializing and is ready to process commands.

event.status.power_change

Power down detection line shows power change (up or down).

Class	`EVENT`	
Types	**state**	: `STRING`
Permissions	guest	`rw`
	admin	`rw`

This event is generated each time power down detection line toggles.

event.status.sync_block

An antenna controlled by DIO has been blocked.

Class	EVENT		
Types	**dio**	:	INT
Permissions	guest	rw	
	admin	rw	

This event indicates an antenna controlled by DIO has been blocked from transmitting. The DIO number is displayed. This applies to INHIBIT sync mode only.

event.status.sync_signal

Reader received a sync signal.

Class	EVENT		
Types	**total_slots**	:	INT
	mode	:	STRING
	role	:	STRING
	period	:	INT
Permissions	guest	rw	
	admin	rw	

This event indicates the reader received a sync signal. The total slots for the previous sync period is displayed, along with the current sync mode, role and period (in microseconds) between the last sync signals.

event.status.sync_slot

Reader started a sync timeslot.

Class	EVENT		
Types	**slot**	:	INT
Permissions	guest	rw	
	admin	rw	

This event indicates the reader started a sync timeslot. The timeslot number is displayed.

event.status.tag_auth_fail

Reader read tag but failed authentication

Class	EVENT		
Types	**tag_id**	:	STRING
	type	:	STRING
	info	:	STRING
Permissions	guest	rw	
	admin	rw	

This event indicates a tag read but the authentication algorithm failed due to the reason in the info field.

event.status.tag_collision

Reader detected collision between tags during inventory round.

Class	EVENT		
Types	**tag_type**	:	STRING
Permissions	guest	rw	
	admin	rw	

This event indicates the reader detected a collision between tags responding during an inventory round.

event.status.tag_loss_of_signal

Reader read tag but signal loss prevented CRC validation.

Class	`EVENT`		
Types	**raw_data**	:	`STRING`
	type	:	`STRING`
	antenna	:	`INT`
	gen2_timestamps	:	`INT`
	frequency	:	`INT`
	initial_clock_offset	:	`INT`
	rssi	:	`INT`
	demod_result	:	`INT`
	tx_power	:	`INT`
	quality_metric	:	`INT`
Permissions	guest	`rw`	
	admin	`rw`	

This event indicates a tag read but signal loss prevented validation of CRC.

event.status.tag_no_epc

Reader ACK'd tag but no EPC was received.

Class	`EVENT`		
Types	**raw_data**	:	`STRING`
	type	:	`INT`
	antenna	:	`INT`
	gen2_timestamps	:	`INT`
	frequency	:	`INT`
	initial_clock_offset	:	`INT`
	rssi	:	`INT`
	demod_result	:	`INT`
	tx_power	:	`INT`
	quality_metric	:	`INT`
Permissions	guest	`rw`	
	admin	`rw`	

This event indicates a tag's RN16 was ACK'd but no EPC was received from the tag.

event.status.tag_read_bad_crc

Reader read tag but tag CRC did not match.

Class	EVENT		
Types	**raw_data**	:	STRING
	type	:	STRING
	antenna	:	INT
	gen2_timestamps	:	INT
	frequency	:	INT
	initial_clock_offset	:	INT
	rssi	:	INT
	demod_result	:	INT
	tx_power	:	INT
	quality_metric	:	INT
Permissions	guest	rw	
	admin	rw	

This event indicates a tag read but the CRC on the tag data did not match the CRC received by reader.

event.status.tx_active

Reader's transmitter has been enabled/disabled.

Class	EVENT		
Types	**active**	:	INT
Permissions	guest	rw	
	admin	rw	

This event indicates the when the reader's transmitter has been enabled or disabled.

event.status.tx_limit_exceeded

Transmit limit is exceeded.

Class	`EVENT`	
Types		
Permissions	guest	`rw`
	admin	`rw`

If a timeout value is set for the EN302208 subregions, the timeout limit is exceeded while there is no tag in the field, the reader is switched to standby mode.

18.12 EVENT.STATUS.TAG.DB

event.status.tag.db.repeat_count_cleared

Reader completed clearing tag repeat count fields.

Class	`EVENT`	
Types	**result** :	`INT`
Permissions	guest	`rw`
	admin	`rw`

18.13 EVENT.TAG

event.tag.alarm

Tag has generated an alarm.

Class	EVENT		
Types	**tag_id**	:	STRING
	type	:	STRING
	time	:	STRING
	antenna	:	INT
Permissions	guest	rw	
	admin	rw	

This event indicates a tag has generated an alarm. The current supported alarms are EasAlarm which is indicated by type=EASALARM, and BatteryLow Alarm, indicated by `type=BATTERY_LOW_ALARM.`

event.tag.antenna_cross

Tag has crossed an antenna.

Class	EVENT		
Types	**tag_id**	:	STRING
	tid	:	STRING
	type	:	STRING
	time	:	STRING
	antenna	:	STRING
	frequency	:	INT
	rssi	:	INT
Permissions	guest	rw	
	admin	rw	

This event is generated when a tag has crossed in front of an antenna. The particular tag that has been crossed can be found in the antenna field of the event.

18 EVENTS NAMESPACE

event.tag.arrive

Tag has arrived.

Class	EVENT		
Types	**tag_id**	:	STRING
	tid	:	STRING
	user_data	:	STRING
	type	:	STRING
	first	:	STRING
	antenna	:	STRING
Permissions	guest	rw	
	admin	rw	

This event is generated when a tag is detected for the first time. The amount of information present is controlled by `tag.reporting.arrive_fields.`

event.tag.authentication

Tag has been authenticated (post-arrival).

Class	EVENT		
Types	**tag_id**	:	STRING
	tid	:	STRING
	user_data	:	STRING
	type	:	STRING
	first	:	STRING
	antenna	:	STRING
Permissions	guest	rw	
	admin	rw	

This event is generated when a tag is authenticated after the arrival event has already been generated.

18 EVENTS NAMESPACE

event.tag.decrypt

Tag encrypted data has been detected.

Class	EVENT		
Types	**handle**	:	STRING
	data	:	STRING
Permissions	guest	rw	
	admin	rw	

This event indicates a tag encrypted data has been detected.

event.tag.depart

Tag has departed.

Class	EVENT		
Types	**tag_id**	:	STRING
	type	:	STRING
	antenna	:	STRING
	tid	:	STRING
	user_data	:	STRING
	last	:	STRING
	repeat	:	STRING
	rssi	:	STRING
	min_rssi	:	STRING
	max_rssi	:	STRING
Permissions	guest	rw	
	admin	rw	

This event is generated when a previously detected tag is no longer detected. The information provided is controlled by `tag.reporting.depart.`

event.tag.estimate

The estimated tag position between antennas has been reported.

Class	`EVENT`		
Types	**position**	:	`STRING`
	position_confidence	:	`INT`
Permissions	guest	`rw`	
	admin	`rw`	

This event indicates the estimated tag position between two antennas. Part of lane discrimination.

event.tag.raw

Raw data for a tag read, or read attempt

Class	`EVENT`		
Types	**raw_data**	:	`STRING`
	type	:	`INT`
	antenna	:	`INT`
	gen2_timestamps	:	`INT`
	frequency	:	`INT`
	initial_clock_offset	:	`INT`
	rssi	:	`INT`
	quality_metric	:	`STRING`
	demod_result	:	`STRING`
Permissions	guest	–	
	admin	`rw`	

This event is generated each time a read is attempted whether or not it is successful. The return values for `quality_metric`, `gen2_timestamps`, `initial_clock_offset`, and `demod_result` are for internal use only.

event.tag.raw_arrive

Tag has raw arrived.

Class	`EVENT`		
Types	**tag_id**	:	`STRING`
	tid	:	`STRING`
	user_data	:	`STRING`
	type	:	`STRING`
	first	:	`STRING`
	antenna	:	`STRING`
Permissions	guest	`rw`	
	admin	`rw`	

The `raw_arrive` event is used to indicate initial tag processing has begun while waiting for regular arrival to signal operation completion. The amount of information present is controlled by `tag.reporting.raw_arrive_fields.`

event.tag.raw_depart

Tag has raw departed.

Class	`EVENT`		
Types	**tag_id**	:	`STRING`
	type	:	`STRING`
	antenna	:	`STRING`
	tid	:	`STRING`
	user_data	:	`STRING`
	last	:	`STRING`
	repeat	:	`STRING`
Permissions	guest	`rw`	
	admin	`rw`	

This event is generated when a previously detected tag is no longer detected. The information provided is controlled by `tag.reporting.depart.` The difference between `raw_depart` and depart is that a `raw_depart` event always occurs after a tag is no longer detected regardless of whether or not filtering or security has allowed the arrive event.

event.tag.report

Tag has been detected.

Class	EVENT		
Types	**tag_id**	:	STRING
	tid	:	STRING
	user_data	:	STRING
	user_data_offset	:	STRING
	type	:	STRING
	time	:	STRING
	antenna	:	INT
	frequency	:	INT
	rssi	:	INT
	tx_power	:	INT
	initial_clock_offset	:	INT
	gen2_timestamps	:	INT
	quality_metric	:	INT
Permissions	guest	rw	
	admin	rw	

This event indicates a tag has been detected. The amount of information presented is controlled by `tag.reporting.report.`

event.tag.scan_tags_complete

The `tag.db.scan_tags()` command has completed.

Class	EVENT	
Types		
Permissions	guest	rw
	admin	rw

This event indicated that a `tag.db.scan_tags()` command has completed.

event.tag.security

Tag security information.

Class	EVENT		
Types	**tag_id**	:	STRING
	tid	:	STRING
	rssi	:	INT
	tid_authentic	:	STRING
	pw_authentic	:	STRING
	packet_counter	:	INT
	tag_type_index	:	INT
	info	:	STRING
Permissions	guest	rw	
	admin	rw	

This event shows the in-depth information related to tag security processing.

18.14 EVENT.WARNING

event.warning.antenna

Reader detected a warning condition on a specific antenna.

Class	EVENT		
Types	**id**	:	INT
	time	:	STRING
	text	:	STRING
Permissions	guest	rw	
	admin	rw	

This event indicates that the reader detected a warning condition on a specific antenna port.

18 EVENTS NAMESPACE

event.warning.communication

Reader detected a warning condition with communications.

Class	`EVENT`		
Types	**id**	:	`INT`
	time	:	`STRING`
	text	:	`STRING`
Permissions	guest	`rw`	
	admin	`rw`	

The reader has detected communication warning condition.

event.warning.configuration

Reader detected a configuration warning condition.

Class	`EVENT`		
Types	**id**	:	`INT`
	time	:	`STRING`
	text	:	`STRING`
Permissions	guest	`rw`	
	admin	`rw`	

This event indicates that the reader detected a warning condition during configuration.

event.warning.file_handling

Reader detected a warning condition while operating on a file.

Class	`EVENT`		
Types	**id**	:	`INT`
	time	:	`STRING`
	text	:	`STRING`
Permissions	guest	`rw`	
	admin	`rw`	

This event indicates that the reader detected an warning condition while operating on a file.

event.warning.hw

Reader detected warning condition in hardware health monitoring.

Class	`EVENT`		
Types	**id**	:	`INT`
	time	:	`STRING`
	text	:	`STRING`
Permissions	guest	`rw`	
	admin	`rw`	

The reader has detected a warning condition during hardware health monitoring.

event.warning.license

Reader license is expiring or has expired

Class	`EVENT`		
Types	**id**	:	`INT`
	time	:	`STRING`
	text	:	`STRING`
Permissions	guest	`rw`	
	admin	`rw`	

These events go into the warning log as a warning about license status changes.

event.warning.radio

The reader detected a warning in the radio module.

Class	EVENT		
Types	**id**	:	INT
	time	:	STRING
	text	:	STRING
Permissions	guest	rw	
	admin	rw	

The reader detected a warning condition in the radio module.

event.warning.sw

Reader detected a warning condition with the software.

Class	EVENT		
Types	**id**	:	INT
	time	:	STRING
	text	:	STRING
Permissions	guest	rw	
	admin	rw	

This event indicates that the reader detected a warning condition with the software.

19 DIAG NAMESPACE

19 DIAG NAMESPACE

19.1 DIAG

diag.clear_error_status

Clears the error status variable

Class	FUNCTION		
Parameters	**timestamp**	:	STRING
Response	``		
Permissions	guest	–	
	admin	x	

This function clears the reader's error status variable. If a timestamp is provided, the status only clears if the most recent cause of the error indication occured at the specified timestamp. (`i.e.` if an error occurred subsequent to the specified timestamp, the error indication will remain intact)

diag.clear_fault_led

Clears the reader's fault LED.

Class	FUNCTION	
Parameters		
Response	BOOL	
Permissions	guest	–
	admin	x

This function causes the reader clear the fault LED.

19 DIAG NAMESPACE

diag.cpu_temp

CPU temperature sensor value.

Class	VAR		
Type	int		
		default value:	0
		min	0
		max	999
Permissions	guest	x	
	admin	x	

This variable returns the current value of the CPU temperature sensor.

The following example gets CPU temperature value.

```
>>> diag.cpu_temp
ok 360
```

diag.error_status

Indicates whether an error or warning has occurred.

Class	VAR	
Type	enum	
	default value: none	
Permissions	guest	r
	admin	r

Indicates whether an error or warning has occurred; status is cleared with `diag.clear_error_status()` function. Used primarily by RMT tool.

19.2 DIAG.ERROR_HANDLER

diag.error_handler.period

Time period at which the error event filter is flushed.

Class	`VAR`	
Type	`int`	
	default value: `60`	
Permissions	guest	–
	admin	`rw`

The longer this time, the less frequently you get events when there are multiple error/warning/info events of the same type.

19.3 DIAG.ERROR_THRESHOLD

diag.error_threshold.period

Time interval between error threshold adjustments.

Class	`VAR`	
Type	`int`	
	default value: `30`	
Permissions	guest	–
	admin	`rw`

The longer this time, the more events can occur before the threshold count is reduced. When this time period expires, each event threshold is incremented by one. When each event occurs, the threshold is decremented by one. Once the threshold reaches zero, the event is processed, action (if any) is taken, and the threshold is reset.

19.4 DIAG.WSD

diag.wsd.log_level

Log level associated with WSD service.

Class	`VAR`	
Type	`enum`	
	default value: `error`	
Permissions	guest	`r`
	admin	`rw`

This variable sets the logging level of WSD Service.

the WSD service will need to be restarted for this new log level to take effect. Valid levels are: error, warning, info, debug

20 INTERPRETING READER LOGS

20 INTERPRETING READER LOGS

When the Neology Toll RFID Reader 7204 generates an error or warning event, these events are written to an error log that can later be viewed using `reader.view_log()` function or the embedded Reader Configuration Tool (RCT).

Each entry in the reader log has the following format:

`time`	The date and time of the event
`event`	The event name. This is usually the top level warning or error. For detailed explanations of the error or warning, refer to the Event Namespace chapter.
`id`	The subevent identifier number. (For Neology use only.)
`text`	The specific subevent. For detailed explanations of the subevent, refer to the specific event in the Event Namespace chapter.
`pid`	The process identifier number.
`file`	The directory path and filename where the error or warning event originated.
`line`	The line of code within the originating file
`info`	A brief description of the event
`action`	the code for the reader action that was taken after the event. (For Neology use only.)

The following is a sample from an actual reader log:

```
time=2009-08-25T13:42:23.027, event.error.sw,id=38,

text='aux_process_failure', pid=1211,
file=ErrorHandler/ErrorHandler.c,

line=170, info='event occurred 1 times in last 54
seconds.', action=0


time=2009-08-25T13:43:03.575, event.error.sw,id=38,

text='aux_process_failure', pid=1211,
file=tag_security/TagSecurity.c,

line=3253, info='Error ending cmd seqence', action=0
```