Interface Control Document Part 1: One and Only Protocol for Smallsats! (OOPS!)

V1.3.3, April 2021

TXMission Inc
30 S. Calle Cesar Chavez, Suite D
Santa Barbara, CA 93103, USA
Tel: +1 805 965 3669

TXMission Ltd
CP House, Otterspool Way
Watford, WD25 8HU, UK
Tel: +44 1923 889 542

https://txmission.com

Contents

1 Introduction

1.1 About this document

This interface control document defines a software protocol for communicating with our **Quest™** onboard and **Connect™** ground station modems, shown in **Figure 1**. A second interface control document (Part 2) defines the hardware interfaces (connector pinouts and electrical and mechanical characteristics).

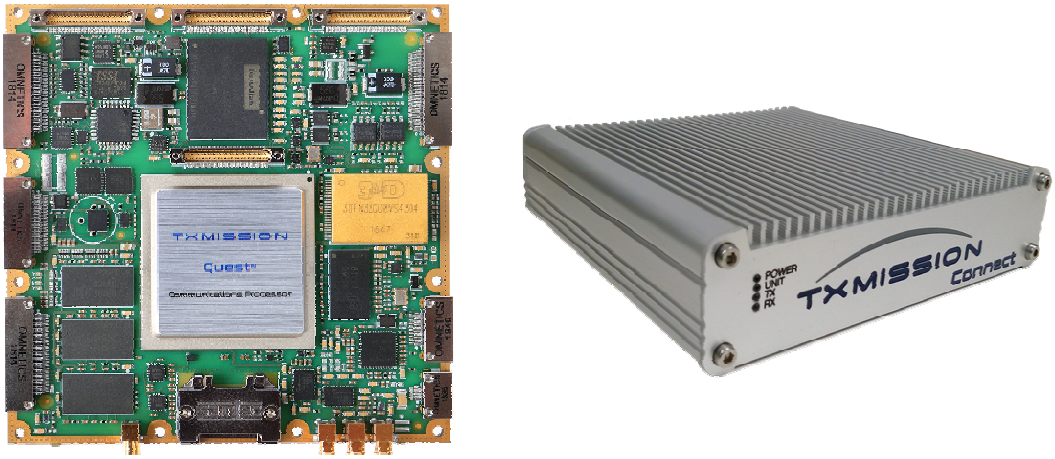

Figure 1: Quest™ Onboard Smallsat Modem and Connect™ Ground Station Modem

Figure 2 shows the OOPS! terminal emulator built into our **MissionSpan™** network control application.

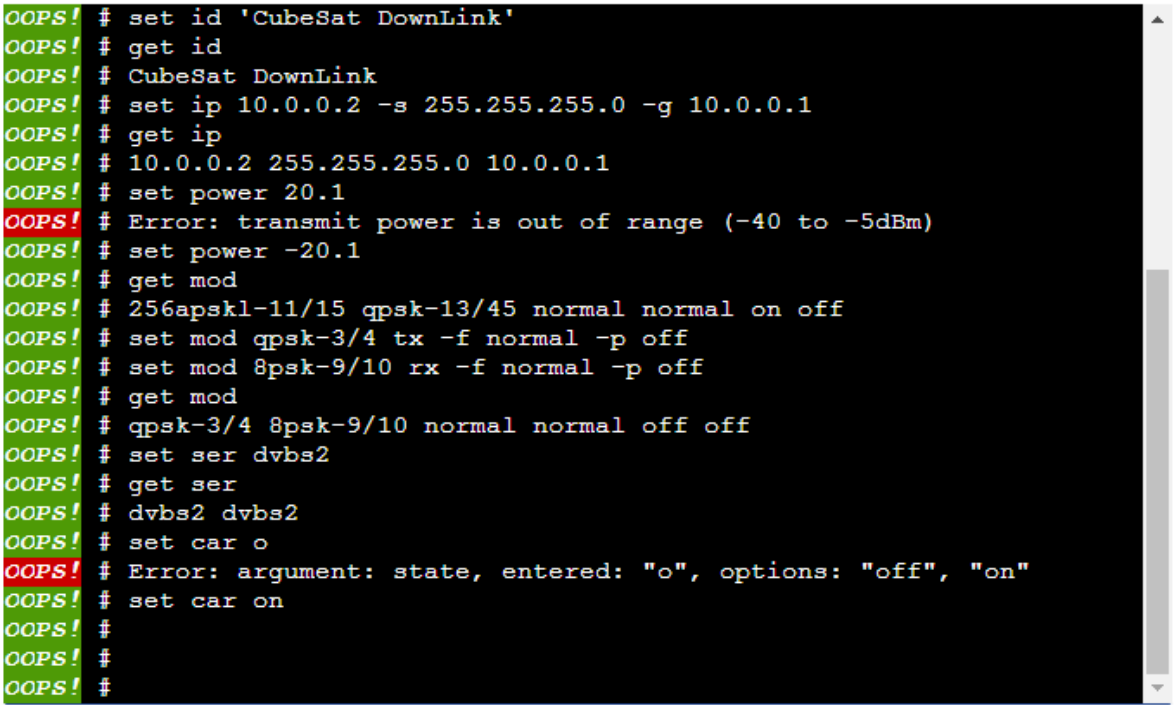

Figure 2: OOPS! Terminal Emulator Feature of MissionSpan™ Network Control Application

The One and Only Protocol for Smallsats! (OOPS!) is an open-source protocol for communicating with low and medium-earth satellites (to which, for convenience, we have assigned the rather arbitrary collective title of 'smallsats'). OOPS! provides an alternative to using the built-in web user interfaces provided by our products. OOPS! was created by TXMission to address the notable shortcomings found in existing space protocols with respect to their ability to be applied to the unique requirements of the New Space industry, which continues to undergo an unprecedented revolution in technology. Following its initial development, OOPS! is now being freely offered to the space community as an open standard. It is anticipated that it will continue to be developed in future under the auspices of an open industry forum that TXMission is committed to helping to set up and support.

The scope of OOPS! is to control and monitor all onboard subsystems and the ground-segment equipment counterparts that are required to implement an end-to-end smallsat communications system. It therefore covers both uplink and downlink, housekeeping data, telemetry data and other requirements. It is suitable for a wide range of applications, including earth observation, telecoms, IOT and 5G.

OOPS! operates at the application layer with respect to both the TCP/IP and OSI models. It is compatible with various transport mechanisms including TCP/IP, CCSDS packet telemetry (specified in CCSDS 102.0-B-4) and CubeSat Space Protocol (CSP) (which is a lightweight alternative to TCP/IP). Internally, TXMission uses OOPS! in conjunction with automated SSH control via TCP/IP as part of our MissionSpan network management system (we take the view that TCP/IP is set to become the dominant smallsat communications technology). We use CCSDS as an optional complementary technology that provides a mechanism for transporting OOPS! and other data from specific onboard sources to specific handler processes on the ground. OOPS! leaves all security and reliability issues to the other layers.

We also use OOPS! with a TCP/IP transport layer that uses the ZeroMQ (0MQ or zmq) sockets library (see https://zeromq.org/). The reason is that it gives us complete control over the socket communications rather than having to rely on the quirks of particular SSH implementations. ZeroMQ is incredibly robust and takes care of typical socket issues, such as retries when packets are not received, reconnecting the client to the server after errors, etc. We use ZeroMQ with node.js and JavaScript (although the ZeroMQ library itself is compiled code for optimum efficiency). We have a full client implementation of OOPS! including a terminal emulator (based on Xterm.js) and client-side parsing of commands (to do up-front checks for format errors prior to transmission) that we make freely available to users who want to create their own modem control applications. This allows users to communicate fully with our modems from their own applications with very little effort.

Much of the value of OOPS! lies in the fact that it is a lightweight protocol that is intuitive to use (being human readable), easy to implement and readily extensible. From the outset it has supported the ability for any organisation to add its own command extensions with minimum formality. While an open forum is still in the process of being formed, TXMission is happy to collate and make public any new OOPS! documentation that is provided to us by third parties.

1.2 Overview

OOPS! is a command line protocol that uses the grammatical conventions enshrined in both the Linux and Windows command line languages and by many others in addition. Newcomers will therefore often find it provides a familiar framework, as its structure and usage tend to resonate with prior experiences of other command languages.

The basics are *<value>* represents a non-optional command line argument that must be provided while *[value]* is an optional argument. Flags (e.g. *-h* or *–help*) are used for command options (often representing Boolean choices) and may or may not require arguments of their own. Commands may support aliases that are shortened forms of the full command name. An example is:

set modcod <value> [path] [--frame <size>] [--pilots <state>]

where *set modcod* is the command name, *value* represents the modcod selection and must be present, *path*, which indicates what to apply the modcod setting to, is optional, *frame* and *pilots* are optional flags but if present must be accompanied by an argument that represents what value these functions are to be set to. Various abbreviations and defaults will typically be specified in the full command description.

Although OOPS! is a command line protocol that lends itself to being typed in by a user via a telnet or SSH session (or through a serial interface) it can also be incorporated into fully automated operational and test management systems.

OOPS! currently recognises the following smallsat subsystems:

- eps: electrical power system
- radio: communications system (typically a combination of a satellite modem/Software Defined Radio and associated RF/antenna).

- adcs: attitude determination and control system
- obc: onboard computer (sometimes referred to as a flight computer)
- payload: payload system

The above terms are used as command prefixes to identify which subsystem a command relates to. For example, a smallsat can be made to generate a carrier by sending it the following command:

radio.set carrier on

The OOPS! command handler remembers the last subsystem addressed and the prefix is therefore not required on subsequent consecutive commands to the same subsystem. (The default prior to any prefix being used for the first time is to send the commands to the radio subsystem.) For example, if at a later point we want to switch the carrier off, we could send the abbreviated command:

set carrier off

If, however, we had addressed commands to other subsystems in the intervening period then we would have to issue the full form of the command for it to be recognised:

radio.set carrier off

Note that commands are always in lower case. Each command must be terminated with a new line even when control is automated. The definition of a new line varies with operating system, being a line feed (‘\n’) in Linux and a carriage return and line feed (‘\r\n’) in Windows.

Commands are executed at the time they are received. There is a concept of configurations, which are conceptually like files, although in our case they represent records in a database. A configuration record represents all the information needed to fully configure a radio. When the user loads a named configuration then it is applied to the radio hardware. A copy of the configuration is made as the *current* configuration. Any changes that the user subsequently makes via the command line are made to this copy. When the radio is reset or power cycled then it loads this *current* configuration from the database, complete with any changes that the user had made since loading the named configuration. The user must explicitly save these changes back to the original named configuration or a different configuration, if required, otherwise the changes would be lost if another configuration were to be loaded.

It is recommended with respect to the radio that the transmit carrier is muted during reconfiguration of the modulator in order to prevent the potential for a carrier with undefined characteristics being generated. This is not necessary when loading a configuration since all settings are applied at the same time and the carrier will be muted during this process if necessary.

Every command will generate a response and the user should wait for a response before sending the next command. Responses end with a new line followed by '# ', which indicates that the equipment is ready to receive another command. If a command were to fail then an error message will be generated that starts with the text '*Error:*' (which is then terminated by the usual new line and '# '). An example of a command with its response is as follows:

 # get carrier
off
#

This indicates that the modulator's carrier is currently muted.

As mentioned, configurations can be stored (in non-volatile memory) for later use. The *save* command is used to create a new configuration where each setting for the radio is set to whatever the *current* configuration is. For a brand-new radio, this represents the factory defaults (there is a permanent *factory* configuration on each modem to allow copies to be easily made as the starting point for new configurations). If the named configuration already exists then *save* will overwrite it. For example:

 # save myconfig
 # set service dvbs2

creates a configuration file called *myconfig* from the *current* configuration and then sets the service to DVB-S2 in the *current* configuration.

A stored configuration can be enabled as the current active configuration using the *load* command:

 # load myconfig

Configurations can be viewed and deleted as follows:

 # list
myconfig1
myconfig2
myconfig3
 # del myconfig1
#

A list of all subsystem commands can be displayed by issuing the *help* command:

 # help

2 Radio Commands

This chapter specifies the commands that can be used with respect to the smallsat's radio subsystem and the ground system counterpart. Appendix A (TXMission Radio Performance) specifies limitations in relation to specific TXMission products (such as the maximum valid data rate). These limitations must be observed when entering commands.

2.1 Command Quick Guide

delete <config>
get acm
get alarms
get carrier
get dr [path]
get encrypt
get frequency [path]
get identity
get interface [path]
get ip
get log
get metrics [--mod] [--dem] [--tra] [--deb]
get modcod [path]
get ntp
get power
get rolloff [path]
get service [path]
get sr [path]
get status [path]
get summary
get time
get test [path] [--all] [--prbs] [--bist]
get video
help
list
load <config>
reset [--hw] [--sw] [--log] [--alarms] [--counts]
save <config>
set acm <state>
set carrier <state>
set dr <value> [path]
set encrypt <state> [--pass <password>]
set frequency <value> [path]
set identity <name>
set interface <type> [path]
set ip <address> [--subnet <mask>] [--gateway <gateway>]
set modcod <value> [path] [--frame <size>] [--pilots <state>]
set ntp <state> [--server <address>] [--interval <period>]
set power <value>
set rolloff <value> [path]
set service <mode> [path]
set sr <value> [path]
set test [path] [--loop <state>] [--mode <state>] [--pattern <pattern>]
set time <time> [--date <date>]
set video <state>
spawn <request>

2.2 delete

delete <config>

Alias:
del

Arguments:
config Specifies the name for an existing radio configuration

Deletes the named configuration. See related configuration commands: *list*, *load* and *save*. Configurations are records that are held in an embedded database on each radio.

Examples:

```
# list
  myconfig1
  myconfig2
  myconfig3
# del myconfig1
#
```

2.3 get acm

get acm

Alias:
get ac

Returns the current DVB-S2/S2X ACM state (i.e. whether ACM is on or off). When off, the modem operates in DVB-S2/S2X CCM mode. These modes of operation are explained in the DVB-S2/S2X standards (EN 302 307-1 and EN 302 307-2). When on, the transmit modcod is selected based on the Es/No value reported by the receiver, assuming a feedback channel is available between the receiver and transmitter. ACM cannot be used if this feedback channel does not exist, as there is nothing to base the choice of modcod on. ACM should be switched on in both the transmitter and receiver. The ACM feedback mechanism is proprietary; the message format is available on request.

Examples:

```
# get acm
  on
# get acm
  off
#
```

2.4 get alarms

get alarms

Alias:
get al

Returns the radio's log of timestamped historic and current alarms.

Examples:

```
# get alarms
  Mon, 14 Dec 2020 14:46:02 GMT: Warning: Intermittent decoder errors
  Mon, 14 Dec 2020 14:46:09 GMT: Warning: Demodulator unlocked
  Mon, 14 Dec 2020 14:46:33 GMT: Alarm status: Modem OK
#
```

2.5 get carrier

get carrier

Alias:
get car

Returns the transmit status of the radio's carrier (i.e. whether the carrier is on or off).

Examples:

```
# get carrier
  on
# get car
  off
#
```

2.6 get dr

get dr [path]

Alias:
get d

Arguments:
path *tx, rx*

Returns the current transmit and receive data rates in Mbps. If neither *tx* nor *rx* is specified then the command returns the transmit followed by the receive data rate.

Examples:

 # get dr
10.222123 4.134780
 # get dr tx
10.222123
#

2.7 get encrypt

get encrypt

Alias:
get enc

Returns the status of the radio's AES-256 encryption function (i.e. whether encryption of TCP/IP traffic between the transmitter and receiver is on or off).

Examples:

 # get encryption
on
 # get enc
off
#

2.8 get frequency

get frequency [path]

Alias:
get freq

Arguments:
path *tx, rx*

Returns the current transmit and/or receive carrier center frequency in MHz. If neither *tx* nor *rx* is specified then the command returns the transmit frequency followed by the receive frequency.

Examples:

 # get frequency
2325.000000 2225.000000
 # get freq tx
2325.000000
#

2.9 get identity

get identity

Alias:
get id

Returns the name associated with the radio that is used to identify it to the user.

Examples:

```
# get id
  CubeSat Downlink 1
#
```

2.10 get interface

get interface [path]

Alias:
get int

Arguments:

path	*tx, rx*	[Note: the *path* argument is reserved for future use; currently, it is assumed that the same interface is used for both transmit and receive.]

Returns the physical interface associated with the transmission and reception of data in the radio. Supported interfaces: *eth, usb, lvds, spacewire, rs485, sdi, uart, can, i2c, spi*.

Examples:

```
# get interface
  eth
#
```

2.11 get ip

get ip

Alias:
get i

Gets the IP address for the radio's monitor and control interface, including the subnet mask and gateway.

Examples:

 # set ip 10.0.0.2 -s 255.255.255.0 -g 10.0.0.1
 # get ip
10.0.0.2 255.255.255.0 10.0.0.1
#

2.12 get log

get log

Alias:
get lo

Returns the contents of the radio's system log. This contains a description of the 3,000 most recent system events such as alarms, reconfiguration, power cycles, user login, etc. Entries are timestamped.

Examples:

 # get log
Wed Sep 09 2020 11:32:21 GMT+0100 (British Summer Time): Loaded configuration: current
Wed Sep 09 2020 11:32:21 GMT+0100 (British Summer Time): Retrieved user data
Wed Sep 09 2020 11:32:21 GMT+0100 (British Summer Time): Retrieved configuration data
Wed Sep 09 2020 11:32:21 GMT+0100 (British Summer Time): Retrieved test configuration: current
Wed, 09 Sep 2020 10:32:21 GMT: GET /login 200
#

2.13 get metrics

get metrics [--mod] [--dem] [--tra] [--deb]

Alias:
get met

Options:

-m, --mod	Reserved for future use
-d, --dem	Reserved for future use
-t, --tra	Reserved for future use
-d, --deb	Reserved for future use

Returns a set of TCP/IP packet metrics associated with the radio's traffic shaper. These correspond to the information shown on the *Traffic Metrics* web page.

The default operation of the command is to return the metrics associated with the set of Diffserv DSCP classes supported by the radio's traffic shaper. The metrics returned for

each class are: number of packets transmitted, number of bytes transmitted, number of packets dropped and the number of over-limit packets. The over-limit number indicates how many times packets have been delayed. The last line gives the totals for each of these metrics. The metrics can be reset either by reconfiguring the traffic shaper or by using the OOPS! *reset* command.

Examples:

 # get metrics
CS0 18301 28390358 1768101 0
CS1 18780 28395360 1768344 0
CS2 18761 28395233 1768241 0
CS3 150368 2233692 3886 0
CS4 17122 29381652 1662539 0
CS5 17092 29380278 1663498 0
CS6 1972 2448123 5118 0
CS7 1970 2447835 5126 0
AF11 18654 20289164 1273201 0
AF12 18652 20288902 1273108 0
AF13 1008 2776841 3127 0
AF21 996 2774553 3120 0
AF22 18704 28966310 1774318 0
AF23 0 0 0 0
AF31 1564 2235339 3883 0
AF32 0 0 0 0
AF33 17664 28397693 1645195 0
AF41 0 0 0 0
AF42 1453 2276731 3125 0
AF43 0 0 0 0
EF 18676 29565214 1786551 0
LE 21 22034 12589 10225
Default 0 0 1293 0
Total 341758 288665312 16424363 10225
#

2.14 get modcod

get modcod [path]

Alias:
get mod

Arguments:
path *tx, rx*

Returns the modcod associated with the transmit and/or receive path of the radio. It also returns the frame size (*normal*, *short*) and pilots (*off*, *on*) settings in the case of DVB-S2/S2X services. If neither *tx* nor *rx* is specified then the command returns the transmit followed by the receive modcod, then the transmit/receive frame sizes and then the transmit/receive pilots settings.

For a list of the returned modcod values and a description of frame sizes and pilots, see *set modcod*.

Examples:

```
# get modcod rx
  qpsk-13/45 short off
# get modcod tx
  256apskl-11/15 normal on
# get modcod
  256apskl-11/15 qpsk-13/45 normal short on off
#
```

2.15 get ntp

get ntp

Alias:
get n

Returns the NTP server IP address and the last UTC time message fetched from the NTP server. This requires the NTP client to have been previously enabled on the radio. To get an up-to-date server time, prior to issuing the *get ntp* command, switch the NTP client off and on again, as this initiates an immediate NTP server request.

Example:

```
# set ntp off
# set ntp on
# get ntp
   NTP server 216.239.35.0; NTP last time from server: 2021-04-13T13:22:50.458Z
#
```

2.16 get power

get power

Alias:
get pow

Returns the power level of the transmitted carrier in dBm.

Examples:

```
# get power
  -22.3
#
```

2.17 get rolloff

get rolloff [path]

Alias:
get rol
get roll

Arguments:
path *tx, rx*

Returns the current transmit and receive carrier roll-off factors associated with the transmit and/or receive carriers in the radio. If neither *tx* nor *rx* is specified then the command returns the transmit followed by the receive roll-off.

Examples:

```
# get rolloff
  35 25
# get rol tx
  35
#
```

2.18 get service

get service [path]

Alias:
get ser

Arguments:
path *tx, rx*

Returns the service being provided by the transmit and/or receive paths in the radio. If neither *tx* nor *rx* is specified then the command returns the transmit followed by the receive service.

Examples:

```
# get service tx
  dvbs2x
# get ser
  dvbs2x dvbs2
#
```

2.19 get sr

get sr [path]

Alias:
get s

Arguments:
path *tx, rx*

Returns the current transmit and receive symbol rates in Msps. If neither *tx* nor *rx* is specified then the command returns the transmit followed by the receive symbol rate.

Examples:

```
# get sr
  5.261212 4.134780
# get sr tx
  5.261212
#
```

2.20 get status

get status [path]

Alias:
get sta
get stat

Arguments:
path *tx, rx*

Returns current status information for the transmit and/or receive paths in the radio.

In order, the information that is returned for transmit is: carrier status (*on* or *off*), output power (dBm), data rate (Mbps), symbol rate (Msps), frequency (MHz), CPU loading (%) and temperature (degrees C).

In order, the information that is returned for receive is: Es/No (dB), Eb/No (dB), wanted power (dBm), composite power (dBm), data rate (Mbps), symbol rate (Msps), frequency (MHz), frequency offset (Hz), frame count, frame error count, CPU loading (%) and temperature (degrees C).

If neither *tx* nor *rx* is specified then the command returns: carrier status (*on* or *off*), output power (dBm), transmit data rate (Mbps), transmit symbol rate (Msps), transmit frequency (MHz), receive Es/No (dB), receive Eb/No (dB), receive data rate (Mbps), receive symbol rate (Msps), receive frequency (MHz), receive frequency offset (Hz), DVB-S2/S2X received frame count, DVB-S2/S2X received frame error count, CPU loading (%) and temperature (degrees C).

Examples:

 # get status tx
on -30.9 64.123456 29.457723 2205.244302 45 64
 # get stat rx
10.3 7.8 64.373321 25.448976 2200.066912 13 10134 2 45 64
 # get sta
on -30.9 64.123456 29.457723 2205.244302 10.3 7.8 64.373321 25.448976
2200.066912 13 10134 2 45 64
#

2.21 get summary

get summary

Alias:
get sum

Returns current summary status information for the modem.

In order, the information that is returned is: transmit status (*OK, Off, Muted, No input data*), receive status (*OK, Off, Unlocked, Intermittent errors*), receive Es/No in dB (or *** if the demodulator is unlocked), Ethernet status (*Up, Down, Unknown*), received DVB-S2/S2X frame count (e.g. *144523673839*); received frames error count and modem temperature in degrees C (e.g. 65).

Examples:

 # get summary status
Status: Tx OK, Rx OK; EsNo: 10.91; Ethernet: Up; Frames: 23445738477; Errors: 0; Temp: 68
 # get sum
*Status: Tx Off, Rx Off; EsNo: ***; Ethernet: Down; Frames: 33145883551; Errors: 0; Temp: 67*
#

2.22 get test

get test [path] [--all] [--prbs] [--bist]

Alias:
get tes

Arguments:
path *tx, rx*

Options:
-a, --all Returns all test settings and metrics
-p, --prbs Returns just PRBS BER test metrics

-b, --bist Returns just built-in-self-test results

Returns the test mode settings and test metrics for the transmit and/or receive paths in the radio.

If neither *tx* nor *rx* is specified then the command returns the transmit followed by the receive test settings. In order, the information that is returned for transmit is: RF loopback mode (*on* or *off*), PRBS mode (*on* or *off*), PRBS pattern (*prbs11*, *prbs15*, *prbs20*, *prbs23*). In order, the information that is returned for receive is: RF loopback mode (*on* or *off*), PRBS mode (*on* or *off*), PRBS pattern (*prbs11*, *prbs15*, *prbs20*, *prbs23*). The RF loopback mode is a common status setting for transmit and receive and will therefore have the same value. In order, the information that is returned when no path argument is provided is: RF loopback mode (*on* or *off*), transmit PRBS mode (*on* or *off*), transmit PRBS pattern (*prbs11*, *prbs15*, *prbs20*, *prbs23*), receive PRBS mode (*on* or *off*), receive PRBS pattern (*prbs11*, *prbs15*, *prbs20*, *prbs23*).

The optional flags override the above behaviour to some extent. The *prbs* flag will return just the PRBS BER test results, namely, the number of PRBS bits received, the number of PRBS bit errors, the PRBS BER rate, a PRBS receive sync indicator (0 or 1) and a PRBS receive sync-steady indicator (0 or 1). The sync indicator reflects whether the receiver is currently locked to the transmitter. The sync-steady indicator reflects whether there has been a loss of lock at any time since the test started. The *bist* flag will return just the built-in-self-test results. (Using both flags at the same time will result in one, but not both, sets of results being returned.) The *all* flag will output all test settings and results.

Examples:

```
# get test tx
  off on prbs23
# get test
  on on prbs 11 on prbs11
# get test -a
  on on prbs 11 on prbs11 4356691208 1 0.9981772988852798 1 1 ram ok rom ok soc ok
  rf ok
# get test tx -a
  on on prbs 11 4356691208 1 0.9981772988852798 1 1 ram ok rom ok soc ok
  rf ok
# get test -p
  4356691208 1 0.9981772988852798 1 1
# get test -b
  ram ok rom ok soc ok rf ok
```

2.23 get time

get time

Alias:

get ti

Returns the modem's current time and date. Note that the time is reported as UTC rather than local time.

Example:

```
# get time
  Mon Mar 29 17:57:39 UTC 2021
#
```

2.24 get video

get video

Alias:
get vid

Returns the status of the radio's H.264/H.265 image/video compression feature (i.e. whether compression is on or off).

Examples:

```
# get video
  on
# get vid
  off
#
```

2.25 help

help

Alias:
h

This command displays a full list of the available commands pertaining to the radio subsystem.

Examples:

```
# help
delete <config>
get alarms
get carrier
get dr [path]
…
```

2.26 list

list

Alias:
ls

Lists all the configurations stored on the radio. See related configuration commands: *delete*, *load* and *save*.

Examples:

```
# list
  current, Factory configuration, myconfig1,  myconfig2, myconfig3
#
```

2.27 load

load <config>

Alias:
l

Arguments:

config	Specifies the name of an existing radio configuration

Reconfigures the radio using the contents of the named configuration. Note that any commands that are subsequently issued are applied to the current configuration, not the configuration that was loaded. See related configuration commands: *delete*, *list* and *save*.

Examples:

```
# load myconfig
#
```

2.28 reset

reset [--hw] [--sw] [--log] [--alarms] [--counts]

Alias:
res

Options:

-h, --hw	Performs a hard reset of the radio (via hardware reset signal)
-s, --sw	Performs a soft reset of the radio (reboots operating system)
-l, --log	Clears the radio's system log
-h, --alarms	Clears the radio's alarm log
-c, --counts	Clears all the radio's counts (frames, frame errors, PRBS bits,

traffic shaper metrics, etc.)

At least one option must be provided.

A soft reset performs a soft reset of the radio, which pulls a reset signal to restart the radio.

A software reset restarts the software only, whereas a hard reset goes through a similar sequence to a power cycle.

Various counts (as shown on the *Status*, *Test* and *Traffic Metrics* web pages) can be reset to zero.

2.29 save

save <config>

Alias:
sav

Arguments:

config	Specifies the name for the radio configuration (new or existing). Valid names are as per Linux filenames. (However, the configuration is held as a record in a database and not as a file and therefore there is no concept of nested folders.) If the configuration exists, it will be overwritten

Saves the current radio configuration to a named configuration. TXMission implements support for multiple radio configurations through an embedded database that can hold up to 10,000 named configurations. Configurations can be considered as virtual files and can be manipulated in a similar way to files within the OOPS! non-volatile database.

Examples:

 # save myconfig
 # set service dvbs2 tx
 # save myconfig

This creates a new configuration or overwrites the existing configuration if *myconfig* already exists. The contents of *myconfig* are a copy of the *current* configuration that is active on the radio at the time the *save* command is issued. The current transmit path service is then set to DVB-S2 and this change is then saved back to *myconfig*. See related configuration commands: *delete*, *list* and *load*.

2.30 set acm

set acm <state>

Alias:
set ac

Arguments:
state *off, on*

Switches DVB-S2/S2X ACM on or off. When off, the modem operates in DVB-S2/S2X CCM mode. These modes of operation are explained in the DVB-S2/S2X standards (EN 302 307-1 and EN 302 307-2). When on, the transmit modcod is selected based on the Es/No value reported by the receiver, assuming a feedback channel is available between the receiver and transmitter. ACM cannot be used if this feedback channel does not exist, as there is nothing to base the choice of modcod on. ACM should be switched on in both the transmitter and receiver. The ACM feedback mechanism is proprietary; the message format is available on request.

Examples:

 # set acm on
#

2.31 set carrier

set carrier <state>

Alias:
set car

Arguments:
state *off, on*

Switches the transmit carrier off or on.

Examples:

 # set car on

2.32 set dr

set dr <value> [path]

Alias:
set d

Arguments:

value	Specifies the transmit and/or receive data rate in Mbps
path	*tx, rx*

Sets the transmit and receive data rates in Mbps. If neither *tx* nor *rx* is specified then the command sets both.

The range of supported data (and symbol) rates is hardware, service and modcod specific. Note that there is a fixed relationship between data rate and symbol rate and changing one will change the other regardless of what prior commands have been issued. (Some users will prefer to work in symbols and others in bits.)

Examples:

```
# set dr 453.234478
# get dr
  453.234478 453.234478
#
```

2.33 set encrypt

set encrypt <state> [--pass <password>]

Alias:

set enc

Arguments:

state	*off, on*
password	*variable length string (consisting of any alphanumeric characters)*

Options:

-p, --pass	Sets the password used to generate the encryption key

Switches AES-256 encryption off or on and allows a passphrase to be set.

Encryption is an optional export-controlled feature.

The passphrase is used to generate an encryption key. The passphrase must be the same on both radios that are exchanging TCP/IP data. All TCP/IP traffic is securely encrypted in a VPN tunnel between the radios when encryption is on.

Caution should be taken when enabling and disabling encryption, and when changing the encryption key, to make sure that communications with the remote radio are not lost due to incompatible settings between the two radios.

Examples:

```
# set encrypt on -p txmission
# set enc off
#
```

2.34 set frequency

set frequency <value> [path]

Alias:
set freq

Arguments:

value	Specifies the transmit or receive carrier center frequency in MHz. The range of supported frequencies is dependent on the RF capabilities of the radio
path	*tx, rx*

Sets the transmit and receive carrier center frequencies in MHz. If neither *tx* nor *rx* is specified then the command sets both.

Examples:

```
# set freq 2300.000000
# get freq
  2300 2300
```

2.35 set identity

set identity <name>

Alias:
set id

Arguments:

name	A string that specifies a unique name for the equipment that is meaningful to the user (such as its location or service description)

Sets a name for the radio that can be used to identify it to the user.

Examples:

```
# set id “CubeSat 1 downlink”
# get id
  CubeSat 1 downlink
# set id hub
# get id
  hub
#
```

2.36 set interface

set interface <type> [path]

Alias:
set int

Arguments:

type	Specifies the physical interface to be used by the radio when transmitting or receiving data. Supported interfaces: *eth, usb, lvds, spacewire, rs485, sdi, uart, can, i2c, spi*
path	*tx, rx* [Note: the *path* argument is reserved for future use; currently, it is assumed that the same interface is used for both transmit and receive.]

Sets the physical interface associated with the transmission and reception of data in the radio.

Examples:

 # set interface eth
#

2.37 set ip

set ip <address> [--subnet <mask>] [--gateway <gateway>]

Alias:
set i

Arguments:

address	An IPv4 address (e.g. 192.168.123.55) representing the monitor and control Ethernet port of the radio
mask	A subnet mask (e.g. 255.255.0.0)
address	A gateway address (e.g. 192.168.123.1)

Options:

-s, --subnet	Sets the monitor and control Ethernet port subnet mask
-g, -gw, --gateway	Sets the monitor and control Ethernet port gateway address

Sets the IP address for the radio's monitor and control interface. The factory default is 192.168.70.100 with a subnet mask of 255.255.255.0 and with no default gateway (i.e. gateway is 0.0.0.0). Note that the use of CIDR (Classless Inter-Domain Routing) notation (e.g. /24) is not supported when defining addresses and subnets.

Our onboard radios support only a single Ethernet connection, which is used for both monitor/control and traffic. Our ground modems support two Ethernet connections (one for monitor and control and the other for traffic). The radio operates as a Layer 2 switch and does not allow the traffic interface to be assigned an address. This does not prevent

the common practice of having the monitor and control function on a separate subnet to the network traffic in order to create traffic separation.

Setting a monitor and control address (or using the default address) is required for accessing the built-in web server and for issuing OOPS! commands over Ethernet from a command line application.

Although the command only accepts IPv4 addresses, all IPv6 packets on the network will be correctly bridged through the radio.

Examples:

 # set ip 10.0.0.2 -s 255.255.255.0 -g 10.0.0.1
 # set ip 192.168.3.25
#

2.38 set modcod

set modcod <value> [path] [--frame <size>] [--pilots <state>]

Alias:
set mod

Arguments:

value	Specifies the transmit or receive modulation and FEC rate
path	*tx, rx*
size	*normal, short*
state	*off, on*

Options:

-f, --frame	Sets the DVB-S2/S2X frame size (*normal* or *short*). Normal frames cause higher latency but are more spectrally efficient and require less power than short frames
-p, --pilots	Sets the DVB-S2/S2X pilots setting (*off* or *on*). Pilots allow a carrier to be received lower into the noise but reduce spectrally efficiency

If neither *tx* nor *rx* is specified then the command is applied to both paths. Note that it does not matter in which order *set service* and *set modcod* commands are applied. In other words, the modcod does not have to be valid for the current service type when setting the modcod and vice versa, thereby avoiding being unnecessarily prescriptive in terms of how the user should configure the radio. (Of course, the modem will not operate correctly until all the configuration settings are compatible with each other.)

For DVB-S2X normal frames, the valid values are *qpsk-13/45, qpsk-9/20, qpsk-11/20, 8psk-23/36, 8psk-25/36, 8psk-13/18, 8apskl-5/9, 8apskl-26/45, 16apsk-26/45, 16apsk-3/5, 16apsk-28/45, 16apsk-23/36, 16apsk-25/36, 16apsk-13/18, 16apsk-7/9, 16apsk-77/90, 16apskl-5/9, 16apskl-8/15, 16apskl-1/2, 16apskl-3/5, 16apskl-2/3, 32apsk-32/45, 32apsk-11/15, 32apsk-7/9, 32apskl-2/3, 64apsk-11/15, 64apsk-7/9, 64apsk-4/5, 64apsk-5/6, 64apskl-32/45, 128apsk-3/4, 128apsk-7/9, 256apsk-32/45, 256apsk-3/4, 256apskl-29/45, 256apskl-2/3, 256apskl-31/45, 256apskl-11/15.*

For DVB-S2X short frames, the valid values are *qpsk-11/45, qpsk-4/15, qpsk-14/45, qpsk-7/15, qpsk-8/15, qpsk-32/45, 8psk-7/15, 8psk-8/15, 8psk-26/45, 8psk-32/45, 16apsk-7/15, 16apsk-8/15, 16apsk-26/45, 16apsk-3/5, 16apsk-32/45, 32apsk-2/3, 32apsk-32/45.*

For DVB-S2 normal frames, the valid values are *qpsk-1/4, qpsk-1/3, qpsk-2/5, qpsk-1/2, qpsk-3/5, qpsk-2/3, qpsk-3/4, qpsk-4/5, qpsk-5/6, qpsk-8/9, qpsk-9/10, 8psk-3/5, 8psk-2/3, 8psk-3/4, 8psk-5/6, 8psk-8/9, 8psk-9/10, 16apsk-2/3, 16apsk-3/4, 16apsk-4/5, 16apsk-5/6, 16apsk-8/9, 16apsk-9/10, 32apsk-3/4, 32apsk-4/5, 32apsk-5/6, 32apsk-8/9, 32apsk-9/10*.

For DVB-S2 short frames, the valid values are *qpsk-1/4, qpsk-1/3, qpsk-2/5, qpsk-1/2, qpsk-3/5, qpsk-2/3, qpsk-3/4, qpsk-4/5, qpsk-5/6, qpsk-8/9, 8psk-3/5, 8psk-2/3, 8psk-3/4, 8psk-5/6, 8psk-8/9, 16apsk-2/3, 16apsk-3/4, 16apsk-4/5, 16apsk-5/6, 16apsk-8/9, 32apsk-3/4, 32apsk-4/5, 32apsk-5/6, 32apsk-8/9*.

For CCSDS Viterbi/Reed-Solomon, the valid values are *bpsk-1/2, bpsk-2/3, bpsk-3/4, bpsk-5/6, bpsk-7/8, qpsk-1/2, qpsk-2/3, qpsk-3/4, qpsk-5/6, qpsk-7/8, oqpsk-1/2, oqpsk-2/3, oqpsk-3/4, oqpsk-5/6, oqpsk-7/8.*

Examples:

```
# set modcod qpsk-1/2 tx
# set modcod 8psk-3/4 rx -f short -p off
# set mod 256apskl-11/15 –frame normal --pilots on
# get mod
  256apskl-11/15 256apskl-11/15 normal normal on on
#
```

2.39 set ntp

set ntp <state> [--server <address>] [--interval <period>] [--offset <local>]

Alias:

set n

Arguments:

state	*off, on*
address	The IPv4 address (e.g., 216.239.35.0) of the NTP server
period	The number of minutes (in the range 1 to 1440) between NTP server requests. This is reserved for future use
offset	The number of minutes (in the range 1 to 1440) to offset the received NTP server time by to set a local time. This is reserved for future use

Options:

-s, --server	Sets the NTP server IP address (the default is 216.239.35.0)
-i, --interval	Sets the NTP server polling period (the default is 6 hours)
-o, --offset	Reserved for future use

Uses the Network Time Protocol (NTP) to synchronize the time of the radio to an external server. The radio supports an NTP client that polls an NTP server represented by an IP

address. The default IP address of 216.239.35.0 represents the server at time.google.com. The network to which the receiving ground station modem is attached will need to have access to an NTP server, either directly or via the internet. Instructions on how to share a computer internet connection with a modem are provided in the description of NTP in the *Quest™ and Connect™ Modem User Interface Manual.*

An NTP request is made as soon as NTP is switched on and thereafter at a fixed rate of every six hours. Note that OOPS! *set time* commands can be issued even when NTP is active, although it is not recommended. The NTP protocol uses UTC time.

Examples:

 # set ntp on -s 132.163.96.5 -i 240
 # set ntp off
#

2.40 set power

set power <value>

Alias:
set pow

Arguments:

value	Specifies the transmit power level at the output of the radio prior to any RF amplification, in the range -40.0 to -5.0dBm

Sets the transmit carrier power level for the radio.

Examples:

 # set power -22.3
#

2.41 set rolloff

set rolloff <value> [path]

Alias:
set rol
set roll

Arguments:

value	Specifies the transmit or receive carrier roll-off factor (one of the following values: 5, 10, 15, 20, 25, 35, 40)
path	*tx, rx*

Sets the transmit and receive carrier roll-offs to the given value. If neither *tx* nor *rx* is specified then the command is applied to both paths.

Examples:

 # set rolloff 35
 # get rol
35 35
 # set rol tx 25
#

2.42 set service

set service <mode> [path]

Alias:
set ser

Arguments:

mode	Specifies the transmit or receive service. This is one of: off, *dvbs2*, *dvbs2x*, *ccsds-dvbs2*, *ccsds-dvbs2x*, *ccsds-vit*, *ccsds-vit-rs*
path	*tx, rx*

Sets the transmit and receive services to the given mode. If neither *tx* nor *rx* is specified then the command is applied to both paths. Transmit and receive services are independent of each other and can run different services at the same time.

Examples:

 # set service dvbs2 tx
 # set ser dvbs2x rx
#

2.43 set sr

set sr <value> [path]

Alias:
set s

Arguments:

value	Specifies the transmit and/or receive symbol rate in Msps
path	*tx, rx*

Sets the transmit and receive symbol rates in Msps. If neither *tx* nor *rx* is specified then the command sets both.

The range of supported symbol (and data) rates is hardware, service and modcod specific. Note that there is a fixed relationship between data rate and symbol rate and changing one will change the other regardless of what prior commands have been issued. (Some users will prefer to work in symbols and others in bits.)

Examples:

 # set sr 100.0
 # get sr
100 100
 # set sr 33.123456 rx
#

2.44 set test

set test [path] [--loop <state>] [--mode <state>] [--pattern <pattern>]

Alias:
get tes

Arguments:

path	*tx, rx*
state	*off, on*
pattern	*prbs11, prbs15, prbs20, prbs23*

Options:

-l, --loop	*Sets RF loopback mode (off or on). This loops back the transmit path to receive, causing the modem to receive its own transmitted carrier*
-m, --mode	*Sets the PRBS BER test mode (off or on)*
-p, --pattern	*Sets the PRBS BER test pattern*

This command sets test modes for the transmit and/or receive paths in the radio. If neither *tx* nor *rx* is specified then the command applies the settings to both.

While the PRBS settings are path dependent, the RF loopback mode is strictly independent of either path and therefore the path option being present or absent has no effect with respect to controlling loopback mode.

Examples:

 # set test tx -m on -p prbs20
 # set test rx -m on -p prbs23 -l
 # set test -l off
 # set test -m off
#

2.45 set time

set time <time> [--date <date>]

Alias:
set ti

Arguments:

time	*time in format hh:mm:ss (e.g. 18:30:00)*
date	*date in format 'yyyy-mm-dd'*

Options:

-d, --date	Sets the current date

Sets the radio's time and date.

Examples:

```
# set time 18:00:00 -d 2021-03-29
#
```

2.46 set video

set video <state>

Alias:

set vid

Arguments:

state	*off, on*

Switches H.264/H.265 image/video compression off or on. Detection of the coding standard, frame rate, etc. is automatic.

Examples:

```
# set video on
#
```

2.47 spawn

spawn <request>

Alias:

sp

Arguments:

request	Any valid Linux command supported by the radio

Executes the given Linux operating system command and returns the output from *stdout*. It is important that the command completes.

Examples:

```
# spawn date
  Wed Apr 14 09:44:01 UTC 2021
```

```
# spawn 'netstat -r'
  Kernel IP routing table
  Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
  192.168.70.0    *               255.255.255.0   U         0      0        0 br0
```

3 Appendix A: TXMission Platform Performance

Platform	Parameter	Performance
Quest Hardware Platform A/B	Data rate in DVB-S2/S2X modes	0.050000 to 800.000000Mbps
Quest Hardware Platform A/B	Data rate in Viterbi-RS modes	0.009600 to 50.000000Mbps
Connect Hardware Platform	Data rate in DVB-S2/S2X modes	0.050000 to 800.000000Mbps
Connect Hardware Platform	Data rate in Viterbi-RS modes	0.009600 to 50.000000Mbps
Quest Hardware Platform A/B	Symbol rate in DVB-S2/S2X modes	0.100000 to 119.000000Msps
Quest Hardware Platform A/B	Symbol rate in Viterbi-RS modes	0.009600 to 40.000000Msps
Connect Hardware Platform	Symbol rate in DVB-S2/S2X modes	0.100000 to 119.000000Msps
Connect Hardware Platform	Symbol rate in Viterbi-RS modes	0.009600 to 40.000000Msps
Quest/Connect VHF/UHF/IF/L/S/C-band RF	Frequency range	75.000000 to 6000.000000MHz