FlexRadio

PowerGenius XL

API Documentation

March 2024

Firmware Version: 3.8.8

FlexRadio, Inc.

4616 W. Howard Lane
Suite 8-860
Austin, TX USA 78728

+1 (512) 535-4713

Contents

Overview

This API allows for the configuration and control of the PowerGenius XL amplifier via IP network, enabling the creation of various meters, information retrieval, antenna interlocking, status monitoring, and system keepalive.

All API communications to the amplifier is over a TCP connection on port: 9008

TCP/IP Command Protocol

Command Prologue

Upon connecting to the client port, the PGXL will provide the firmware version installed, followed and optional 'AUTH' phrase if the client connection is initiated from WAN.

This information is provided each time the client connects to the Antenna Genius.

```
V<a.b.c> [ AUTH]
V          = indicates version information
<a.b.c>    = major version number in decimal '.' minor version number in
decimal '.' build version number in decimal
AUTH       = if the client connection is initiated from WAN, the client must
authenticate itself
```

Command Format

Once the connection is established, the client may begin sending commands to the device. Commands follow a general ASCII format. Command format (from client to device):

```
C<seq_number>|command<terminator>
C             = indicates a command
<seq_number>  = sequence number, numeric, 1 to 255
<terminator>  = 0x0D
```

Response Format

All commands receive responses from the device. At least one response will be sent for each command received by the client. The response is tied to the command by the sequence number supplied by the client. This sequence number will be echoed back to the client in the response. The client should check the hex response to determine if the command issued was successful. A zero (0) value indicates success. Any other value represents a failure of the command to execute. The response value is unique and provides insight into the failure mode. It is recommended that any decisions that are made by the client based on a response should use the hexadecimal response to make the decisions. Response Format (from device to client):

```
R<seq_number>|<hex_response>|<message>
<seq_number>   = numeric, up to 32-bits -- echoed from the command sent from
the client
<hex response> = ASCII hexadecimal number
```

```
<message>      = response value for parsing
```

Status Format

Objects in the device will send out status when they are changed. To find out about objects, the client must have subscribed to the status for the object. Status messages are asynchronous and can be sent to the client at any time. Status Format (from device to client):

```
S<0>|<message>
```

Command and Control API

Amplifier Creation (FLEX Amplifier API)

- **Command**: `amplifier create`
- **Parameters**:
  - `ip` - The IP address of the amplifier.
  - `port` - The network port number for command and control.
  - `model` - The amplifier model identifier.
  - `serial_num` - The serial number of the amplifier.
  - `ant` - Defines the connection map between the FLEX ANT1/2 ports and the PGXL A/B ports.

```
Example:
amplifier create ip=192.168.0.14 port=9008 model=PowerGeniusXL serial_num=2-50/18-0005 ant=ANT1:PORTA,ANT2:PORTB

Note: PGXL serial numbers are inconsistently formatted.
```

Meter Creation (FLEX Meter API)

- **Command**: `meter create`
- **Parameters**:
  - `name` - Name identifier for the meter.
  - `type` - Type of meter (AMP).
  - `min` - Minimum value of the meter reading.
  - `max` - Maximum value of the meter reading.
  - `units` - Units of the meter reading (DBM, DB, AMPS, TEMP_C).

```
•   Example: (Note the PGXL currently (v3.8.8) creates five meters. SSDR
    currently uses three (FWD, RL to calc SWR, and PA TEMP). The DRV meter is
    a recent (v3.8 addition), reporting a drive level against a calibration
    data.

•   meter create name=FWD type=AMP min=30.0 max=63.01 units=DBM
•   meter create name=RL type=AMP min=34.0 max=60.0 units=DB
•   meter create name=DRV type=AMP min=10.0 max=50.00 units=DBM
•   meter create name=ID type=AMP min=0.0 max=70.0 units=AMPS
•   meter create name=TEMP type=AMP min=0.0 max=100.0 units=TEMP_C
```

Interlock Creation (FLEX Interlock API)

- **Command**: `interlock create`
- **Parameters**:
  - `type` - Type of the interlock (AMP). *Note: The value set by the PGXL should be changed to PGXL or more appropriate identifier.*
  - `valid_antennas` - List of valid antennas for interlocking.
  - name - Identifier for the interlock setup.
  - `serial` - Serial number associated with the interlock.

Keepalive Enable (FLEX Amplifier API)

- **Command**: `keepalive enable`
- **Description**: Enables the keepalive functionality to maintain the connection alive. The amplifier will periodically send a ping command to the radio which feeds a watchdog timer.

```
•   Example:
    interlock create type=AMP valid_antennas=ANT1,ANT2 name=PG-XL serial=2-
    50/18-0005
```

Interlock Disable (FLEX Interlock API)

- **Command**: `interlock disable`
- **Parameters**:
  - 3 - The ID of the interlock to disable.

Ping (FLEX Amplifier API)

- **Command**: `ping`
- **Description**: Checks the connectivity with the amplifier.

Message (FLEX API)

- **Command**: `message`
- **Parameters**:
  - `severity` - The severity level of the message (info, warning, error).
  - `code` - A unique code identifying the message.
  - `message` - The text message to be logged or displayed.

```
Example:
message severity=info code=000001 "PGXL is ready for transmissions."
```

Discovery and Connect Process.

When “paired” with a radio, the PGXL will open the FLEX discovery port and filter the payloads of those UDP packets for a serial number that matches the paired value. The PGXL then connects to the radio as a non-gui client, parsing the initial state dump from the radio.

The amplifier then registers itself as an “Amplifier” using the FLEX Amplifier API. The firmware then queries the radio for any remaining gaps in state, subscribes to or creates the object(s) it needs to track for operation and metering.

Meter Creation

The PGXL uses the FLEX Metering protocol and process described here:
Metering Protocol · flexradio/smartsdr-api-docs Wiki (github.com)

PTT and Frequency (Band) Data

Frequency or band data is parsed from data returned asynchronously from a paired FLEX transceiver via a subscription to the radios slice subsystem. Note it is possible to subscribe to slice changes from up to two radios.

The amplifier will track the TX flag and use that where it can prepare the amplifier for a transmission for a slice frequency. If no TX flag is set, the amplifier will not switch bands, but will use a just-in-time mechanism. Note the amplifier will not enable PTT via the LAN mechanism if paired with a FLEX transceiver.

If hard-wired PTT is in use, the amplifier will follow the external PTT signal. Note that there is PTT in and out on the amplifier. A PTT signal asserted at the PTT in port will not be reflected to the PTT out until the amplifier is ready to accept RF.

Band data can come from multiple sources.

1) Over TCP when paired with a FLEX 6000 transceiver
2) CAT/CIV serial
3) BCD input
4) Pin2Band input

Please refer to the PGXL user guide for specific behaviors related to both PTT and Band data.

PowerGeniusXL Configuration and Status API

Overview

This API facilitates communication between the PowerGeniusXL amplifier and a windows application for configuration and status monitoring. Commands are issued to the amplifier and responses are received back, providing near-enough real-time data and control.

API Commands and Responses

Retrieve Amplifier Status

- **Command**: `status`
- **Response**: Returns the current state of the amplifier with detailed status and meter information.

Read Amplifier Setup

- **Command**: `setup read`
- **Response**: Provides the current basic configuration information including nickname, fan mode, MeFFA state and LED intensity.

Read Interface Configuration

- **Command**: `ifconf read`
- **Response**: Returns network configuration details such as IP address, netmask, gateway, and DHCP status.

Read CAT Radio Configuration

- **Command**: `catradio read={A|B}`
- **Response**: Provides the current configuration of the CAT radio for the specified slice (A or B).

Read FlexRadio Configuration

- **Command**: `flexradio read={A|B}`
- **Response**: Returns the FlexRadio configuration details for the specified slice (A or B).

Set Interface Configuration

- **Command**: `ifconf address={ip} netmask={netmask} gateway={gateway} dhcp={true|false}`
- **Response**: Confirms the set network configuration.

Set CAT Radio Configuration

- **Command**: `catradio ampslice={A|B} active={0|1} type={None} baud={baudrate} parity={N} stopbits={1} civ={code}`
- **Response**: 0
  Confirms the CAT serial port radio configuration for the specified slice (A or B).

Set FlexRadio Configuration

- **Command**: `flexradio ampslice={A|B} serial={serial} txant={ANT1|ANT2} ptt={LAN} active={1}`
- **Response**: 0
  Confirms the FlexRadio configuration for the specified slice (A or B).

Save Configuration

- **Command**: save
- **Asynchronous Status**: Indicates the amplifier is rebooting after saving the configuration.

API Usage Examples

To retrieve the current status of the amplifier:

```
C0|status
R0|state=STANDBY ... meffa=OFF
```

To read the current network interface configuration:

```
C0|ifconf read
R0|address=192.168.0.14 ... gateway-status=192.168.0.1
```

To configure FlexRadio for port A:

```
C0|flexradio ampslice=A serial=1718-9765-6601-2203 txant=ANT1 ptt=LAN
active=1
R0|serial=1718-9765-6601-2203 txant=ANT1 ptt=LAN active=1
```

To save the current configuration and reboot the amplifier:

```
save

Asynchronously reported:
S|state=REBOOT
```