Extorr Firmware V0.11 API

Table of Contents

Introduction to the Extorr RGA Firmware

Here is the current state of the project to make the Extorr XT series RGAs respond to simple RS-232 commands and return data to any application. In essence, you initialize (download a software file into) the CCU and then it will respond to ASCII commands to sweep and return data on the serial line. We loosely refer to this software as firmware. We have made a C# demo application that shows how to do this. It will download and start the CCU, provide a graphical interface to control scanning and a console window to display the ASCII sent and received. It will do sweeps and trends, the complete source for the demo (.cs files), makefile for the demo, and the firmware (qpbox.l2) are in the .zip attached. It would be expected at this point that someone wishing to use this firmware would be able to tell more by the demo source code than the following explanation which is only an outline.

How to use the Demo, RGA.exe

1. Run the program rga.exe. Go to the Operating Tab.
2. In the Communications box, select the Port of the CCU.
3. In the Communications box, select the Speed to download the firmware at.
4. Once a Port and Speed are selected, Click the Boot button to download the firmware.
5. A dialog box comes up and shows you the download status. It goes away when the boot is finished.
6. If the boot is successful, a connection will automatically open at the selected Speed from the boot.
7. You can click on Refresh to verify that you are talking to the box. The firmware has a default value set for each parameter at this point.
8. Once the firmware has been started, the calibration data should be loaded into the RGA. (See Loading Configuration Data for details)

Once the calibration data has been loaded, try performing a mass sweep. To do this, turn the filament on by checking the “Filament On” box. Move to the Plot tab and press the “sweep” button. A plot will appear with values that the RGA is reporting for the sweep.

The console window on the operating tab will show what is (sent) and (recv) on the serial line as the CCU is operating. It is pretty simple, you can set parameters, ask for data such as the total pressure, do a “sweep” from LowMass to HighMass (data is stored in the CCU) and do a “stream” to get that data.

bootfirm.exe

bootfirm.exe is a simple program which boots the firmware onto the RGA using the same method as rga.exe. bootfirm.exe allows a user to create their own controlling program without having to worry about booting the firmware themselves, as they can let bootfirm.exe handle that portion of the job.

bootfirm.exe communicates through a command-line interface and must be given 3 arguments.

Usage: $ bootfirm comport baudrate filepath

Arguments:
comport – the comport that is connected to the RGA to use for serial communication.
baudrate – the baud rate used to boot the firmware.
(baudrate must be one of 9600, 19200, 38400, 57600, 115200)
filepath – the path to the qpbox.l2 firmware file with respect to the location of bootfirm.exe

Example

```
bootfirm com3 115200 ..\qpbox\qpbox.l2
The following serial ports were found:
COM8 COM9 COM3
firmware image: ..\qpbox\qpbox.l2
Waiting for byte: 0xac
Downloading level-2 boot loader chunks...
{Init=1}
{PacNum=1}
{PacNum=2}
…
{PacNum=40}
ok:all channels cleared     // starting message from RGA
Finished
```

To view available comports, executing bootfirm without any arguments will report a list of serial ports found.

How to Download the Firmware

This section provides an in-depth guide on how to properly download the firmware to the RGA. The process can be broken down into 5 steps (see Overview). An implementation of this process has been provided in the C# source code files download.cs and bootfirm.cs. The source code has been provided to act as an example for users to be able to refer to and to solidify their understanding of the process. It is recommended to follow along this section by referring to download.cs to see examples of this process. (specifically see function doDownload())

Overview

1. Reset the RGA CCU
2. Download the boot packets
3. (optionally) increase the baud rate
4. Download the rest of the firmware
5. Start the firmware

Reset the RGA CCU

In order to ensure that the RGA is in a state to accept the firmware, it must be reset.

In order to reset the RGA, it must be sent 0x00 bytes (all 0's) for roughly one second.

One simple way to achieve this is to send 1000 bytes of 0x00 at a baud rate of 9600. (see function resetQpBox() for example)

Once the CCU has been reset, it will respond with a byte 0xAC every few seconds at 9600 Baud. This means that the CCU is ready to accept the boot packets.

Download the boot packets

After receiving character 0xAC, immediately send the boot record. *Do not wait longer than 2 seconds to send the boot record, as doing so creates the potential for errors.*

The boot record consists of exactly the first 2560 bytes of the firmware file (qpbox.l2).
{Init1="~~~~~~~~~`/Pwegf&&^BB^,&Eetc00"} 1st part boot packet ~1200 bytes
{Init2="~~~~~~gdjvbuJI&gcggf77etc00"} 1st + 2nd boot packet = 2560 bytes

The CCU will reply **{Init=1}** now boot code is running

(optionally) increase the baud rate

After receiving **{Init=1}** the CCU is now capable of accepting commands to change its baud rate.

Example: **{PacNum=1,Baud=115200}**

The CCU will reply **{PacNum=1}** at the original baud rate, then immediately change to the new baud rate for next packet. At this point, the rest of the firmware is ready to be downloaded

To avoid errors, do not wait longer than 2 seconds to send the baud packet. This is due to the fact that, the CCU will automatically reset if it does not receive any information for more than a few seconds on the serial line during this process.

Download the rest of the firmware

The firmware is loaded 1 packet at a time. (This is simply the remainder of the firmware file qpbox.l2.) Each packet looks like:

{PacNum=2,Index=0,Points=1000,InitCCU="/P_`/Pwegf&&^BB^,&EBOnBp9d&,r/cF4rG!3n8

CCU replies **{PacNum=2}**

And so on until the last firmware packet.

{PacNum=XXX,Index=XXX,Points=XXX,InitCCU="/P_`/Pwegf&&^BB^,&EBnd&,r/cF4rG!3n8 "}

CCU replies **{PacNum=XXX}**

To avoid errors, do not wait longer than 2 seconds to send the firmware packets. This is due to the fact that, the CCU will automatically reset if it does not receive any information for more than a few seconds on the serial line during this process.

Start the firmware

Once all packets have been sent, start the firmware.

To start the firmware send: **{Go}**

ok:all channels cleared reply from CCU, firmware is running.

The firmware starts running at the same baud rate that it was loaded at. For example, if the baud rate had been increased to 115200 for the loading process, the firmware would respond with the running acknowledgement **ok:all channels cleared** at 115200 baud

*Once the firmware has been started, the calibration data should be loaded into the RGA. (See Loading Configuration Data for details)

Loading Configuration Data

Once the firmware has been started, it is crucial to load the configuration data into the RGA.

The calibration data is RGA specific and is determined by the serial number of the specific unit.

The configuration file from the factory with the data is named **sn*XXXX*_factory_cal.cfg**, where the field ***XXXX*** stands for the specific RGA's serial number (not necessarily 4 digits).

In the configuration file lies a field **<CalibrationParameters />**. Here you will find a list of the calibration parameters specific to your RGA.

The firmware starts with general ballpark values for these parameters to get the unit running, however these parameters must be set by the user once the firmware is loaded for optimal performance.

Using the **set** command as described in the Instruction Set section, set each variable according to the calibration parameters in the .cfg file.

The RGA/firmware does not remember these configuration parameters once reset; however, your controlling program is free to save these variables to be loaded after each startup using the **set** command.

*The "debug" variable in the CalibrationParameters has been deprecated and can be ignored.

Firmware Interaction

Overview

All communication to and from the CCU is line based. The CCU reads whole lines, and always sends whole lines based off of the line feed character (‘\n’). A user program is free to read character at a time or line at a time, but the CCU itself is line-based and most programs will probably deal with it by doing line at a time input and output

The CCU receives lines from the controlling application, and does not echo the characters it receives, and it does not provide any facility to allow the user to edit a line.

Generally speaking, the system revolves around a set of variables that the CCU defines. Examples are LowMass, HighMass, SamplesPerAmu, IonizerAmps, etc. When the controlling program sets the value of LowMass, the CCU remembers it and uses it for the low end of mass sweeps. The controlling program can also request the value of a variable, and it will be reported over the serial line. All variables can be read, and some variables can be set as well. Any attempt to access an unknown variable, or set the value of a read-only variable, results in an error message as described below. Some variables are simply remembered when set, and other variables cause various side-effects to occur. When a variable's value is queried, some variables just report the remembered value. Others do a real-time calculation to report the value. For example, the SupplyVolts variable reports the measured power supply voltage when its value is queried. It is an error to try to set this variable. Similarly, some variables cause live side-effects to occur when set. For example, the if you set the variable FilamentEmissionMa to 2.1, the CCU causes 2.1 milliamps to be the set point for the filament emission.

The specific format of the lines that are sent to the CCU and received from the CCU is designed to support both very simple interactions and more complicated interactions, in a syntactically consistent way. For both sending and receiving, lines consist of colon-separated fields, as described below.

Command Syntax

*For a complete list of commands, arguments and exceptions, see the Instruction Set section.

All Commands sent to the CCU follow the same general format:
Command:ArgName:ArgValue

Depending on the command, there may be a range of allowed arguments and values.

The CCU will respond with an error if there is a problem with a given command.

Examples

symbols

Some commands do not have any arguments.
An example of such command is the **symbols** command, which returns a list of all symbols.

get

Some commands require exactly 1 argument.
An example of such command is the **get** command.

Sending **get** with no arguments will result in the CCU responding with an error:

```
(send) get
(recv) error: too few fields in get command
```

To use the **get** command properly, use a colon to delineate the command call from its argument:

```
(send) get:ScanSpeed
(recv) ok:ScanSpeed:24.00
```

set

Some commands require exactly 2 arguments.
An example of such command is the **set** command.

Sending **set** with too few arguments will result in the CCU responding with an error:

```
(send) set:ScanSpeed
(recv) error: too few fields in set command
```

Attempting to **set** a symbol that does not exist will result in the CCU responding with an error:

```
(send) set:FooBar:32
(recv) error:symbol 'FooBar' unknown
```

To use the **set** command properly, a symbol and value must be provided:

```
(send) set:ScanSpeed:20
(recv) ok:ScanSpeed:20.00
```

channel

Some commands do not require any arguments, but can be optionally given arguments.
An example of such command is the **channel** command

Sending **channel** will result in the CCU responding with a list of all 12 channels and their current settings:

```
(send) channel
(recv) ok:channel:0:amu:0.0:dwell:42.00:enabled:0
(recv) ok:channel:1:amu:0.0:dwell:42.00:enabled:0
…
(recv) ok:channel:11:amu:0.0:dwell:42.00:enabled:0
```

However, **channel** can also accept up to 3 additional arguments in order to set specific channels:

```
(send) channel:4:amu:18:dwell:21:enabled:1
(recv) ok:channel:4:amu:18.00:dwell:21.00:enabled:1
```

Responses from the CCU

A list of all current responses from the CCU, and the CCU reserves the right to add more in the future. It is a good idea to write your controlling program in such a way that it gracefully deals with prefixes from the CCU that it does not know about (perhaps by ignoring them).

This list does not describe all possible permutations of data after a response, as this typically depends on the command used to initiate the response. For more information on the exact responses for each command, see the Instruction Set section.

ok:

Reports data or changes made from a specific command given

error:

Reports an error

The specific error is described in text in the field after the error prefix. This text is free-formatted, but is guaranteed to not contain colon characters, and it will all be on one line.

The CCU is free to emit error messages at any time, either in response to your input or as a result of some serious failure that it must report. The set of error messages is anticipated to grow over time as the system evolves, and is open-ended.

inf:

An informational report on a particular variable.

When you try to set a variable and the value cannot be set, in addition to reporting an error as described previously, the CCU guarantees to send the current actual value via this informational report, after the error message. For example, if you send:

(send) set:LowMass:500

the CCU will respond as in

(recv) error: value must be in the range [1..310]
(recv) inf:LowMass:1

BeginStream:

Denotes the beginning of a stream of sweep data.

s10:

Denotes the beginning of a sample of sweep data in base 10 encoding of the data.

s16:

Denotes the beginning of a sample of sweep data in base 16 encoding of the data.

s64:

Denotes the beginning of a sample of sweep data in base 64 encoding of the data.

EndStream

Denotes the end of a stream of sweep data

BeginTrend:

Denotes the beginning of a stream of trend data.

t10:

Denotes the beginning of a sample of trend data in base 10 encoding of the data.

t16:

Denotes the beginning of a sample of trend data in base 16 encoding of the data.

t64:

Denotes the beginning of a sample of trend data in base 64 encoding of the data.

EndTrend

Denotes the end of a stream of trend data

Changing Baud Rate

In order to change the baud rate, the **stop** command should be sent before the actual **set:BaudRate** command.

Sending **stop** forces the system into the idle state, ensuring that no data will be transmitted after the **stop** command.

This guarantees that no data will be lost via transmission during the baud rate change.

Once **stop** has been sent, the user is free to send **set:BaudRate** to initiate the baud rate change.

The baud rate value must be one of 9600, 19200, 38400, 57600, 115200.

The CCU will respond with **ok:BaudRate** at the old baud rate, then immediately change to the requested baud rate.

Once **ok:BaudRate** has been received, the user should change their own baud rate to the requested baud rate.

Example:

```
(send) stop
(send) set:BaudRate:115200
(recv) ok:BaudRate:115200       // received at old baud rate, now switch to new baud rate
```

Functionality

The RGA has two main functionalities:

- To perform a mass sweep

Or

- To perform a mass trend

Sweep Mode

In sweep mode, the RGA performs a mass sweep starting from **LowMass** going to **HighMass**, scanning at **ScanSpeed** samples/sec, and taking **SamplesPerAmu** samples/Amu.

```
(send) sweep
(recv) inf:FirstSweep:1
(recv) inf:LastSweep:1
```

The **sweep** command takes an optional argument **count**.

sweep:**count**:*value* - You can specify a count of sweeps to do. If you do not specify a count, the CCU will sweep continuously until stopped or until another sweep is commanded. The sweep data stored can be retrieved using the FirstSweep and LastSweep variables.

The sweep data can be retrieved by using the **stream** command
stream:**sweep**:*value* - Here *value* specifies a sweep number. That sweep number is streamed. If it is no longer available, an error is emitted.

The **stream** command takes two additional optional arguments, **to** and **from** which the user can use to specify a certain range of the sweep to report. By default, **to** and **from** are the original **LowMass** and **HighMass** used to collect the sweep data.

The RGA will respond to a stream command with **BeginStream**, indicating data about the stream that is about to happen.

Example:

```
(send) stream:sweep:3
(recv) BeginStream:LowMass:1:HighMass:45:SamplesPerAmu:6:sweep:3
```

Stream data is sent using the following format:

s*Encoding*:*sample number*:*data*

An example stream response following the above **BeginStream** response:

```
(recv) s10:0:2.275e-13
(recv) s10:1:2.457e-13
…
(recv) s10:269:2.305e-13
EndStream
```

Using the variables **LowMass**, **HighMass**, and **SamplesPerAmu**, a user can determine which Amu a particular sample is from using the **sample number** field response.

In this example case, the number of samples from this stream goes from 0 to 269, totaling 270 samples. This number of 270 was computed by (**HighMass – LowMass + 1) * SamplesPerAmu**
(45 – 1 + 1) * 6 = 270 total samples.

A few more details:

- The CCU saves sweep data in a ring buffer, using the sweep_data[] array for storage. At any time the sweeps from FirstSweep to LastSweep will be in memory, and can be streamed using the stream command and specifying the sweep number.

- When you start a sweep with different sweep parameters, the old data is discarded. If you start a sweep with the same parameters, the new data augments the old.

- The data is stored in a ring buffer, with the oldest data dropping off. The sweep numbers increase regardless.

- The AutoStream variable controls whether the CCU will initiate streams of its own accord. By default, this is set to 1. When auto streaming, the CCU checks during low-priority thread idle time whether new sweep data is available. If so, it starts to stream the most recently-started stream.

Full Example

Use RGA.exe to follow along. The GUI is very intuitive, simply connect to the RGA as described in the [How to Download the Firmware]() section, set sweep and operating parameters accordingly for your application, and press the "sweep" button on the plot tab. A mass sweep will show up, and you can look back at the operating tab to see the data that is streaming from the RGA. You can change how the data is being reported by changing parameters like **Encoding** and **SamplesPerLine**, although base 10 encoding and 1 sample per line is typically the easiest to interpret and the speed will suffice for most applications, the option to change those parameters is present.

```
(send) symbols                    // RGA.exe sends symbols on connect
(recv) ok:LowMass:1
(recv) ok:HighMass:45
(recv) ok:SamplesPerAmu:6
(recv) ok:ScanSpeed:24.00
(recv) ok:AutoZero:0
(recv) ok:AutoStream:1
(recv) ok:Filament:1
(recv) ok:MultiplierVolts:0
(recv) ok:FilamentEmissionMa:2.000
(recv) ok:ElectronVolts:70
(recv) ok:Focus1Volts:-20
```

```
(recv) ok:SamplesPerLine:1
(recv) ok:Encoding:10
(recv) ok:BaudRate:115200
(recv) ok:DegasTimer:0
(recv) ok:GroundVolts:2.562e-2
(recv) ok:ReferenceVolts:2.510
(recv) ok:PiraniTorr:1.536e-3
(recv) ok:PiraniVolts:-3.283e-1
(recv) ok:PiraniOhms:1156.
(recv) ok:PiraniTempVolts:-1.468e-1
(recv) ok:SupplyVolts:23.96
(recv) ok:QuadrupoleDegC:37.79
(recv) ok:InteriorDegC:43.46
(recv) ok:IonizerVolts:1.000e-1
(recv) ok:IonizerAmps:3.000e-1
(recv) ok:IonizerOhms:3.333e-1
(recv) ok:RfAmpVolts:0.0
(recv) ok:SourceGrid1Ma:4.184e-4
(recv) ok:SourceGrid2Ma:1.245e-3
(recv) ok:FilamentDacCoarse:3003
(recv) ok:FilamentDacFine:2047
(recv) ok:FilamentPowerPct:5.208
(recv) ok:FbPlus:0.0
(recv) ok:FbMinus:0.0
(recv) ok:Focus1FB:-20.01
(recv) ok:PiraniCorrVolts:-1.473e-1
(recv) ok:RepellerVolts:-68.00
(recv) ok:PressureAmps:-1.255e-12
(recv) ok:PressureTorr:-1.255e-10
(recv) ok:FilamentStatus:1
(recv) ok:DegasMa:3.671e-1
(recv) ok:IsIdle:1
(recv) ok:LastSweep:0
(recv) ok:FirstSweep:0
(recv) ok:SerialNumber:133
(recv) ok:ModelNumber:300
(recv) ok:PiraniZero:3.260e-1
(recv) ok:Pirani1ATM:2.325
(recv) ok:SwSettleTicks:10
(recv) ok:RfSettleTicks:50
(recv) ok:TotalOffset:2000
(recv) ok:PartialOffset:2000
(recv) ok:LowCalMass:1
(recv) ok:LowCalResolution:615
```

```
(recv) ok:LowCalIonEnergy:5.000
(recv) ok:LowCalPosition:3.700e-1
(recv) ok:HighCalMass:300
(recv) ok:HighCalResolution:1800
(recv) ok:HighCalIonEnergy:5.000
(recv) ok:HighCalPosition:1.000e-1
(recv) ok:TotalCapPf:10.00
(recv) ok:PartialCapPf:3.000
(recv) ok:TotalSensitivity:10.00
(recv) ok:PartialSensitivity:6.000e-1
(recv) ok:VersionMajor:0
(recv) ok:VersionMinor:11
// At this point, calibration data from your specific .cfg file should be loaded before proceeding
(send) set:Filament:1      // turn on filament
(recv) ok:Filament:1
(send) set:HighMass:20             // using gui to set HighMass to 20
(recv) ok:HighMass:20
(send) sweep                       // press "sweep" button on Plot tab
(recv) inf:FirstSweep:1
(recv) inf:LastSweep:1
(recv) BeginStream:LowMass:1:HighMass:20:SamplesPerAmu:6:sweep:1
(recv) s10:0:7.502e-14
(recv) s10:1:8.644e-14
(recv) s10:2:8.811e-14
(recv) s10:3:1.010e-13
(recv) s10:4:1.005e-13
(recv) s10:5:1.049e-13
(recv) s10:6:1.032e-13
(recv) s10:7:9.260e-14
(recv) s10:8:8.811e-14
(recv) s10:9:1.137e-13
(recv) s10:10:8.890e-14
(recv) s10:11:1.110e-13
(recv) s10:12:1.129e-13
(recv) s10:13:1.006e-13
(recv) s10:14:1.033e-13
(recv) s10:15:1.125e-13
(recv) s10:16:9.523e-14
(recv) s10:17:1.147e-13
(recv) s10:18:8.926e-14
(recv) s10:19:1.110e-13
(recv) s10:20:1.103e-13
(recv) s10:21:1.118e-13
(recv) s10:22:1.076e-13
```

```
(recv) s10:23:1.238e-13
(recv) s10:24:1.001e-13
(recv) s10:25:9.224e-14
(recv) s10:26:9.479e-14
(recv) s10:27:9.400e-14
(recv) s10:28:1.048e-13
(recv) s10:29:9.510e-14
(recv) s10:30:7.541e-14
(recv) s10:31:1.204e-13
(recv) s10:32:1.260e-13
(recv) s10:33:1.118e-13
(recv) s10:34:7.321e-14
(recv) s10:35:1.049e-13
(recv) s10:36:9.848e-14
(recv) s10:37:1.074e-13
(recv) s10:38:9.224e-14
(recv) s10:39:1.096e-13
(recv) s10:40:1.135e-13
(recv) s10:41:9.202e-14
(recv) s10:42:1.100e-13
(recv) s10:43:9.361e-14
(recv) s10:44:1.036e-13
(recv) s10:45:1.134e-13
(recv) s10:46:1.077e-13
(recv) s10:47:1.289e-13
(recv) s10:48:1.028e-13
(recv) s10:49:1.059e-13
(recv) s10:50:1.009e-13
(recv) s10:51:9.035e-14
(recv) s10:52:8.095e-14
(recv) s10:53:9.347e-14
(recv) s10:54:9.444e-14
(recv) s10:55:9.813e-14
(recv) s10:56:1.226e-13
(recv) s10:57:9.620e-14
(recv) s10:58:8.411e-14
(recv) s10:59:1.056e-13
(recv) s10:60:1.120e-13
(recv) s10:61:9.541e-14
(recv) s10:62:9.866e-14
(recv) s10:63:8.508e-14
(recv) s10:64:8.288e-14
(recv) s10:65:9.062e-14
(recv) s10:66:1.099e-13
```

```
(recv) s10:67:9.400e-14
(recv) s10:68:7.792e-14
(recv) s10:69:1.053e-13
(recv) s10:70:1.006e-13
(recv) s10:71:9.602e-14
(recv) s10:72:9.057e-14
(recv) s10:73:9.383e-14
(recv) s10:74:1.104e-13
(recv) s10:75:1.035e-13
(recv) s10:76:1.161e-13
(recv) s10:77:9.207e-14
(recv) s10:78:9.633e-14
(recv) s10:79:8.662e-14
(recv) s10:80:1.111e-13
(recv) s10:81:1.012e-13
(recv) s10:82:1.409e-13
(recv) s10:83:1.093e-13
(recv) s10:84:1.043e-13
(recv) s10:85:1.231e-13
(recv) s10:86:1.118e-13
(recv) s10:87:9.774e-14
(recv) s10:88:1.009e-13
(recv) s10:89:1.136e-13
(recv) s10:90:9.945e-14
(recv) s10:91:9.027e-14
(recv) s10:92:9.695e-14
(recv) s10:93:1.019e-13
(recv) s10:94:1.161e-13
(recv) s10:95:1.215e-13
(recv) s10:96:1.028e-13
(recv) s10:97:9.875e-14
(recv) s10:98:1.025e-13
(recv) s10:99:1.075e-13
(recv) s10:100:1.081e-13
(recv) s10:101:9.708e-14
(recv) s10:102:9.673e-14
(recv) s10:103:1.120e-13
(recv) s10:104:1.027e-13
(recv) s10:105:1.022e-13
(recv) s10:106:9.646e-14
(recv) s10:107:8.306e-14
(recv) s10:108:1.114e-13
(recv) s10:109:1.062e-13
(recv) s10:110:1.152e-13
```

```
(recv) s10:111:8.196e-14
(recv) s10:112:1.137e-13
(recv) s10:113:9.176e-14
(recv) s10:114:9.356e-14
(recv) s10:115:9.752e-14
(recv) s10:116:8.934e-14
(recv) s10:117:1.114e-13
(recv) s10:118:1.063e-13
(recv) s10:119:1.012e-13
(recv) EndStream
(recv) inf:FirstSweep:1          // Firmware will start next sweep, sweep:count not specified so it will
(recv) inf:LastSweep:2           // sweep indefinitely until told to stop or another sweep/trend is sent
(recv) BeginStream:LowMass:1:HighMass:20:SamplesPerAmu:6:sweep:2      // AutoStream = 1,
(recv) s10:0:9.299e-14                                                // automatically stream
(recv) s10:1:1.060e-13                                                // next sweep
(recv) s10:2:1.139e-13
(recv) s10:3:1.100e-13
(recv) s10:4:8.767e-14
(recv) s10:5:1.079e-13
(recv) s10:6:1.005e-13
(recv) s10:7:8.715e-14
(recv) s10:8:1.016e-13
(recv) s10:9:8.425e-14
(recv) s10:10:9.857e-14
(recv) s10:11:1.001e-13
(recv) s10:12:8.209e-14
(recv) s10:13:1.075e-13
(recv) s10:14:1.168e-13
(recv) s10:15:1.088e-13
(recv) s10:16:1.057e-13
(recv) s10:17:1.189e-13
(send) stop                         // press “stop”, tell RGA to stop sweeping
(send) set:SamplesPerLine:6         // get 6 samples per line instead of 1
(recv) ok:SamplesPerLine:6
(send) sweep
(recv) inf:FirstSweep:1
(recv) inf:LastSweep:3
(recv) BeginStream:LowMass:1:HighMass:20:SamplesPerAmu:6:sweep:3
// As you can see, there are now 6 samples returned per line instead of 1
(recv) s10:0:1.637e-13:1.786e-13:1.814e-13:2.136e-13:1.670e-13:1.097e-13
(recv) s10:6:1.216e-13:2.404e-13:1.331e-12:1.550e-12:7.834e-13:1.019e-13
(recv) s10:12:1.085e-13:1.162e-13:9.646e-14:1.067e-13:9.826e-14:9.479e-14
(recv) s10:18:1.023e-13:9.800e-14:8.767e-14:9.189e-14:1.021e-13:1.129e-13
(recv) s10:24:1.003e-13:9.844e-14:1.008e-13:1.064e-13:9.492e-14:1.053e-13
```

```
(recv) s10:30:7.506e-14:8.385e-14:1.033e-13:9.822e-14:1.028e-13:1.133e-13
(recv) s10:36:9.150e-14:9.488e-14:8.192e-14:8.060e-14:9.949e-14:1.002e-13
(recv) s10:42:8.139e-14:9.734e-14:1.103e-13:1.029e-13:1.082e-13:9.242e-14
(recv) s10:48:9.906e-14:1.019e-13:8.983e-14:1.019e-13:1.069e-13:9.479e-14
(recv) s10:54:9.154e-14:1.045e-13:1.004e-13:8.354e-14:1.003e-13:8.715e-14
(recv) s10:60:9.703e-14:1.123e-13:1.002e-13:9.769e-14:9.180e-14:1.002e-13
(recv) s10:66:1.045e-13:9.488e-14:1.021e-13:1.061e-13:9.778e-14:1.171e-13
(recv) s10:72:1.044e-13:1.100e-13:1.195e-13:1.104e-13:1.054e-13:1.047e-13
(recv) s10:78:9.501e-14:1.157e-13:1.022e-13:1.142e-13:9.576e-14:1.097e-13
(recv) s10:84:1.267e-13:1.396e-13:1.508e-13:1.697e-13:1.136e-13:1.066e-13
(recv) s10:90:1.050e-13:1.348e-13:1.751e-13:1.513e-13:1.059e-13:8.917e-14
(recv) s10:96:1.119e-13:9.932e-14:1.049e-13:1.060e-13:9.848e-14:9.637e-14
(recv) s10:102:9.422e-14:1.041e-13:9.475e-14:9.405e-14:1.022e-13:9.238e-14
(recv) s10:108:9.136e-14:9.866e-14:1.043e-13:1.137e-13:1.030e-13:1.142e-13
(recv) s10:114:9.721e-14:2.269e-13:2.270e-13:1.654e-13:9.752e-14:9.290e-14
(recv) EndStream
(recv) inf:FirstSweep:1
(recv) inf:LastSweep:4
(recv) BeginStream:LowMass:1:HighMass:20:SamplesPerAmu:6:sweep:4
(recv) s10:0:1.858e-13:2.016e-13:1.771e-13:1.815e-13:1.444e-13:9.312e-14
(recv) s10:6:1.145e-13:2.552e-13:1.318e-12:1.506e-12:7.576e-13:1.104e-13
(send) stop
// Here I use the gui to change to the hex encoding.  It is very simple, just 8 hex characters
// for each 4-byte float.  Unlike base 10, this is completely lossless of precision, and fairly simple.
// Then I click "sweep" again.
(send) set:Encoding:16
(recv) ok:Encoding:16
(send) sweep
(recv) inf:FirstSweep:1
(recv) inf:LastSweep:5
(recv) BeginStream:LowMass:1:HighMass:20:SamplesPerAmu:6:sweep:5
(recv) s16:0:2a34fee6:2a4bce43:2a3c7794:2a60af9e:2a32923a:29d8b847
(recv) s16:6:29e85af7:2a7eeafe:2bb4ea50:2bd4969c:2b531a99:29dfd848
(recv) s16:12:29f9c5a2:29fc3efd:29c11193:29c24195:29ced2ec:29ee319f
(recv) s16:18:29d806f2:29d57444:2a048180:29cf8445:29e44c49:29ecb5a4
(recv) s16:24:29dd12f5:29d8d19d:29c12aea:29db7d9e:29db7d9e:29d5a6f2
(recv) s16:30:29e3e6f6:29dec19d:29f5384e:29f4ec52:29d132ed:29cda2ee
(recv) s16:36:29e0704a:29c35841:29f5d04e:29da019c:29e1219d:29a2c988
(recv) s16:42:29ecb5a4:29a166e0:29d9b59b:2a038429:29f46da7:29e465a0
(recv) s16:48:29e3359d:29d2e19a:29c422ec:29ca4599:29e0704a:29d44446
(recv) s16:54:29cd0aed:29df72f5:29e646f7:29c552ea:29d91d9e:29c32593
(recv) s16:60:29e0bc4a:29f4b9a4:29e679a0:29d3f845:29cf9d97:29f2599f
(recv) s16:66:29df4048:29de5c4a:29b1bae4:29e23849:29aae6e3:29c5eaeb
(recv) s16:72:29cea043:29e03d9f:2a0aa42e:2a037781:29dc2ef3:2a16dcd9
```

```
(recv) s16:78:29f87c50:29bbb991:2a091b80:2a0fe2d9:29c5eaeb:29ee644d
(recv) s16:84:29c4bae9:2a196f88:2a2c3038:2a24918b:29c4ed97:2a023ad4
(recv) s16:90:2a0746d5:2a10e02e:2a1f2032:2a24f6df:29bce993:29d2e19a
(recv) s16:96:29d01c45:29d982f2:2a0b2f84:29ff69a8:29db4af1:29cd3d97
(recv) s16:102:29e6def7:29c58593:29f0aafa:29e4199f:29b9f191:29d4119c
(recv) s16:108:29e62da0:29df599e:29ddf6f3:29cfe998:29b67ae5:29dbfc49
(recv) s16:114:29e3b44b:2a67e8f8:2a81bc29:2a358a3c:29ccbeed:29e8284c
(recv) EndStream
(recv) inf:FirstSweep:1
(recv) inf:LastSweep:6
(recv) BeginStream:LowMass:1:HighMass:20:SamplesPerAmu:6:sweep:6
(recv) s16:0:2a353e3b:2a599c49:2a48a397:2a503599:2a1ca6de:29fd22fb
(send) stop
// Now I use the gui to switch to base 64 encoding. Like the others, there is always an integral number
// of floats reported per line, specified by the SamplesPerLine variable.  Unlike the others,
// that sequence is returned all at once in base64, with no intervening colons. The data appears
// in one field.
(send) set:Encoding:64
(recv) ok:Encoding:64
(send) sweep
(recv) inf:FirstSweep:1
(recv) inf:LastSweep:7
(recv) BeginStream:LowMass:1:HighMass:20:SamplesPerAmu:6:sweep:7
(recv) s64:0:NwgoKuSCMyo9EjwqpFVqKtZ0BCpMKOgp
(recv) s64:6:K4YHKvlEdSpQnrQrA2PPKxbfTSupLfkp
(recv) s64:12:RdDPKfUO6CmfzeMp+Tr0KU407SkqlgEq
(recv) s64:18:0+QAKqJd5ymq2fspSZjkKUIIzin0TuMp
(recv) s64:24:9u7zKZPBuCn4/u0pP5zGKUwo6CmLgaop
(recv) s64:30:8ebQKewixCnyztkpnpXlKU3A6ClO1Oop
(recv) s64:36:QlTOKaTh+CmM3aQpgDUEKp8F4inyCuAp
(recv) s64:42:SlzeKURwzSk5GLUpSfzbKZEZvinsXsop
(recv) s64:48:RpTcKUyQ5yn65vYpmi3TKUjY3ymdDd8p
(recv) s64:54:+D7pKU7U6imZqdQpKmgEKuvGySmcEdQp
(recv) s64:60:j6muKe0KzSn7+vgpTjj1Kffy6Cns0s4p
(recv) s64:66:Q2jQKUWs0ymW8cwpLRYLKvH60inU0gIq
(recv) s64:72:oiH0KU4Q8Sn5Yvgp/AL2KUR01SkqkAgq
(recv) s64:78:REzRKYO1DSpKsO4p9MrkKZtp2SnmirAp
(recv) s64:84:ogHtKYOHECrcih0qLp4RKqKd9SmWfcgp
(recv) s64:90:9/LoKTJ6FSqM0yMq3RgdKkn82yku+gsq
(recv) s64:96:oEHoKUWEzynuktMp9/LoKaAF9SmjkfAp
(recv) s64:102:1foGKvgy9yn1+uUpmMHLKeaesilEcM0p
(recv) s64:108:86LgKTesrSmZqdQp9cLnKaet7ylGgNop
(recv) s64:114:qL38KVLsdCopX4Uqm6tYKkQo1SlLVOEp
(recv) EndStream
```

```
(recv) inf:FirstSweep:1
(recv) inf:LastSweep:8
(recv) BeginStream:LowMass:1:HighMass:20:SamplesPerAmu:6:sweep:8
(recv) s64:0:maVMKpltTiriOigqQnhKKjFWGSpTrPkp
(recv) s64:6:1VAIKoCcgSpl2bUrhH7NK3CmVCuThcUp
(send) stop
// Here I use the gui to specify that 20 floats should be returned on each line.  As you
// can see, below, the lines get bigger.  There is no built-in limit.  The user can specify
// get all the sweep data returned on a single line if he wants.  The QpBox is happy to
// do that.  Or, the user can specify one data point per line.  In whichever format.
(send) set:SamplesPerLine:20
(recv) ok:SamplesPerLine:20
(send) sweep
(recv) inf:FirstSweep:1
(recv) inf:LastSweep:9
(recv) BeginStream:LowMass:1:HighMass:20:SamplesPerAmu:6:sweep:9
(recv)
s64:0:jhEuKvPAYyqOZy8q6xJKKjOaHCqlHf8pgjEPKiu0hCrkB7QrmabOKxZGSyv56usp8/bdKT9gwCmjCeop7
wLQKUzI5Smg3fAp+Qb+KZ6R3Sk=
(recv)
s64:20:/X73KdUUAirsDsIp+GbtKUJAzClCGMgpknW4Ke1Cyyl/nQMqmi3TKepSxSmjDfIpmanUKZUJxClBN
Mcp5jq7KZ0N3ykpvAEqRCTNKUcE2Sk=
(recv)
s64:40:83rcKaAt5ilEEMsp917wKfP23Snm6rIpOPSlKUtI7ylA/Mgp7eLIKaOR8Cn3UuspTfzuKffO7CmXnc8pS
xzjKTlktSmSsb4p6abCKUNo0Ck=
(recv)
s64:60:+RLwKfVy3ylMXPEpRHTVKX9lBSpARMEp7ZbIKZ8F4imiOespPsi/KfIK4ClJTOQpTXToKUcY2yk+yL8p
gIEEKvu6/SlCCM4pTaD0KaKd9Sk=
(recv)
s64:80:ULj+KUGoyynqZscpPpDBKfZu6inVFAIqjt8oKt36GSqbjdUpRHTVKUiw2ynYuAcqizUqKuCiJypFNNop
8uLbKfda6CnX+AIq6lLFKUU02ik=
(recv)
s64:100:6ybMKZX1wSlHZNspnXHWKfN63ClJAOQp81LYKZ014yn2HuIpnvncKZu12Smf4eUp7DbGKeY6uyn
yCuApgHiFKvgGayrrskcqSNjfKem6xCk=
(recv) EndStream
(recv) inf:FirstSweep:1
(recv) inf:LastSweep:10
(recv) BeginStream:LowMass:1:HighMass:20:SamplesPerAmu:6:sweep:10
(recv)
s64:0:kEU3KkSOUCr1gGgq8cRYKt5aHCrbhBEqo136KYEkiCpk0LYrVhLMK+zSTitIsNsp6Fa6KfFG0yn1wucp8
s7ZKZwR1Ck+8MMp81LYKewOwik=
(send) stop
// Here I use the gui to try to set the low mass to be greater than the high mass.  The gui does
// not catch this error, but relies on the QpBox to catch any errors.  You can see here the
// back-and-forth that happens.  First an error is reported by the QpBox, then the QpBox emits
// an informational response to say what the value in question actually is.
(send) set:LowMass:23
(recv) error: LowMass must be less than HighMass
(recv) inf:LowMass:1
```

Trend Mode

In trend mode, the RGA collects sample data from a set of particular AMUs over time, forming a trend.

The trend mode uses the mass table for information on what to sample, and how long to sample for.

To see the mass table, you can send the **channel** command.

```
(send) channel
(recv) ok:channel:1:amu:0.0:dwell:42.00:enabled:0
(recv) ok:channel:2:amu:0.0:dwell:42.00:enabled:0
…
(recv) ok:channel:11:amu:0.0:dwell:42.00:enabled:0
```

By default, all 12 channels are cleared: set to 0 amu, 42ms dwell, and disabled

You can set channels as the following examples show:

```
(send) channel:2:amu:40.25:dwell:167:enabled:1
(recv) ok:channel:2:amu:40.25:dwell:167.00:enabled:1

(send) channel:1:amu:1.8:dwell:21:enabled:1
(recv) ok:channel:1:amu:1.800:dwell:21.00:enabled:1

(send) channel:2:amu:40.25:dwell:50:enabled:1
(recv) ok:channel:2:amu:40.25:dwell:50.00:enabled:1

(send) channel:2
(recv) ok:channel:2:amu:40.25:dwell:50.00:enabled:1
```

Each a channel command is sent, the response always shows the status of the indicated channel. As a special case, you can, as above, query a channel by just giving the channel number and no other parameters.

The mass table now after the following commands:

```
(send) channel
(recv) ok:channel:0:amu:0.0:dwell:42.00:enabled:0
(recv) ok:channel:1:amu:1.800:dwell:21.00:enabled:1
(recv) ok:channel:2:amu:40.25:dwell:50.00:enabled:1
…
(recv) ok:channel:11:amu:0.0:dwell:42.00:enabled:0
```

To do a trend based on the channels that are set up, you send the **trend** command.

```
(send) trend
(recv) inf:FirstSweep:127
(recv) inf:LastSweep:127
```

The **trend** command takes 3 optional arguments:

trend:**coun**t:*value* - You can specify a count of sweeps to do. If you do not specify a count, the CCU will trend continuously until stopped or until another sweep is commanded. The trend data stored can be retrieved using the FirstSweep and LastSweep variables.

trend:**radius**:*value* - sets how many samples to take around target, reports max(samples). Range: [0..3]

radius 0 means only the target amu.
radius 1 means one extra sample on each side (+/- 0.25 amu)
radius 2 means two extra samples on each side (+/- 0.25, +/- 0.50 amu)
radius 3 means three extra samples on each side
The default is radius 2, which makes 5 samples per target altogether.

trend:**size**:*value* - how many datasets to take per trend pass (default is one).

Examples:

```
(send) trend:count:10:size:3
```

This tells the RGA to take 10 sweeps, with radius of 2 (default because **radius** not specified) around the target AMUs, with 3 samples per target AMU.

AutoSweep also works with the trend mode and will automatically initiate streams of data just like sweep mode.

You can also request a specific trend sweep with the **stream** command just like in sweep mode

An example stream of trend data has the following format:

```
(send) stream:sweep:158
(recv) BeginTrend:sweep:158:1.800:40.25
(recv) t10:0:1.457e-12
(recv) t10:1:8.570e-13
(recv) EndTrend
```

Where **BeginTrend:sweep:value:amu1:amu2** denotes the particular sweep number that will be reported, along with the AMUs that will be reported in their respective order. These AMUs are determined by the enabled channels.

The data itself generally follows the sweep format:

t*Encoding*:*sample number*:*data*

Full Example

This example uses AutoStream enabled

```
(send) channel:0:amu:2:enabled:1
(recv) ok:channel:0:amu:2.000:dwell:42.00:enabled:1
(send) channel:1:amu:18:enabled:1
(recv) ok:channel:1:amu:18.00:dwell:42.00:enabled:1
(send) channel:2:amu:44:enabled:1
(recv) ok:channel:2:amu:44.00:dwell:42.00:enabled:1
(send) channel
(recv) ok:channel:0:amu:2.000:dwell:42.00:enabled:1
(recv) ok:channel:1:amu:18.00:dwell:42.00:enabled:1
(recv) ok:channel:2:amu:44.00:dwell:42.00:enabled:1
(recv) ok:channel:3:amu:0.0:dwell:42.00:enabled:0
(recv) ok:channel:4:amu:0.0:dwell:42.00:enabled:0
(recv) ok:channel:5:amu:0.0:dwell:42.00:enabled:0
(recv) ok:channel:6:amu:0.0:dwell:42.00:enabled:0
(recv) ok:channel:7:amu:0.0:dwell:42.00:enabled:0
(recv) ok:channel:8:amu:0.0:dwell:42.00:enabled:0
(recv) ok:channel:9:amu:0.0:dwell:42.00:enabled:0
(recv) ok:channel:10:amu:0.0:dwell:42.00:enabled:0
(recv) ok:channel:11:amu:0.0:dwell:42.00:enabled:0
(send) trend:size:3           // will take 3 samples per channel per sweep
(recv) inf:FirstSweep:166
(recv) inf:LastSweep:166
// autostream starts streaming the trend data automatically
(recv) BeginTrend:sweep:166:2.000:18.00:44.00     // reports mass 2 data first, then 18, then 44
(recv) t10:0:1.787e-12          // mass 2, 1st round
(recv) t10:1:1.307e-13          // mass 18, 1st round
(recv) t10:2:1.514e-13          // mass 44, 1st round
(recv) t10:3:1.794e-12          // mass 2, 2nd round
(recv) t10:4:1.481e-13          // mass 18, 2nd round
(recv) t10:5:1.322e-13          // mass 44, 2nd round
(recv) t10:6:1.801e-12          // mass 2, 3rd round
(recv) t10:7:1.373e-13          // mass 18, 3rd round
(recv) t10:8:1.509e-13          // mass 44, 3rd round
(recv) EndTrend          // denotes end of trend data
(send) stop              // tell RGA to stop trending
(send) set:SamplesPerLine:3            // send 3 samples per line instead of 1
(recv) ok:SamplesPerLine:3
(send) trend:size:3
(recv) inf:FirstSweep:191
(recv) inf:LastSweep:191
(recv) BeginTrend:sweep:191:2.000:18.00:44.00
(recv) t10:0:1.523e-12:1.100e-13:1.224e-13         //mass 2:mass 18:mass 44    1st round
(recv) t10:3:1.519e-12:1.026e-13:1.129e-13         //mass 2:mass 18:mass 44    2nd round
```

```
(recv) t10:6:1.519e-12:1.179e-13:1.147e-13         //mass 2:mass 18:mass 44     3rd round
(recv) EndTrend
```

Degas Mode

To activate degas mode, set the DegasTimer symbol to a positive value in the range [0..600].
Ex: **set:DegasTimer:600** // sets the DegasTimer to 600 seconds

During degas mode the filament current is increased to **DegasMa**(30 mA).

DegasTimer will automatically count down until it reaches 0. You can retrieve the remaining time using the command **get:DegasTimer**

After **DegasTimer** reaches 0, the filament current is automatically set back to **FilamentEmissionMa.**

Degas mode can be turned off at any time by setting **DegasTimer** to 0. (**set:DegasTimer:0)**

In order for degas mode to take effect, the filament must be on. More specifically, ***FilamentStatus*** *must be 3.*

Checksums and Tags

Checksums

All of the preceding examples have assumed that the communication line between the CCU and the controlling terminal is error free. For some applications, this assumption is reasonable for the reliability requirements and the environment of the equipment. But you can also use checksums going both ways to add a level of error detection. If you want to use checksums, you compute a checksum for the line you want to send, and send it along with the line as described below. When the CCU receives such a line, it verifies the checksum before using the line. If there is a checksum error, it doesn't use the line but instead issues an error message.

And critically, if you send a checksum as part of your command, the CCU will include a checksum with any response to that command. In this way, checksums are entirely optional, but they are enforced and responded to in kind when you provide them.

You communicate the checksum as in the following example:

```
(send) set:LowMass:21:ck:1257
```

When the CCU receives this and wants to report success, it will respond with something like

```
(recv) ok:LowMass:21:ck:1143
```

If there is a checksum error, it will report an error message using the normal mechanism, but will do so using a checksum, as in

```
(recv) error: LowMass must be less than HighMass:ck:3824
```

In general, a command line that is sent to the CCU, or data that comes from the CCU, is always a colon-delimited sequence of fields. The checksum occupies the last two field positions. The ck indicates checksum, and the value that follows is the actual checksum, written in base 10. The checksum applies to the line starting with the initial prefix, and continues up to the colon before the ck, the ck itself, and the colon after the ck.

Tags

Using a mechanism similar to checksums, the CCU supports a mode of operation that lets you more easily match specific responses from the CCU to specific queries that you have made. For a command line that you send, you can provide an optional numeric "tag" for that command line. Any response from the CCU to that command will repeat back to you that numeric tag, so your program can match the response to the particular command that generated the response. For example, if you send

```
(send) set:LowMass:17:tag:12345
(send) set:HighMass:45:tag:12345
(send) channel:1:amu:2:enabled:tag:23
```

the system will respond as follows:

```
  (recv) ok:LowMass:17:tag:12345
  (recv) ok:HighMass:45:tag:12345
  (recv) ok:channel:1:amu:2.000:dwell:42.00:enabled:1:tag:23
```

Using tags, your controlling program is free to implement an increasing serial number for commands it sends, or to use it in any other way. The CCU will respond with the tag given, whether it is increasing or not. The tag occupies the last two field positions before the checksum.

Being that you can use the same tag with multiple requests, tags also provide a means of grouping sets of requests and responses.

Combining Tags and Checksums

You can combine tags and checksum if you wish:

```
  (send) set:SamplesPerAmu:18:tag:2:ck:2346
```

and get

```
  (recv) ok:SamplesPerAmu:18:tag:2:ck:2232
```

Symbols

All symbol values can be retrieved by using the **get** command

Certain symbol values can be changed by using the **set** command
Other symbol values cannot be changed by using the **set** command, as they are Read-Only

LowMass

Variable that controls the lowest mass to sample when doing a mass sweep

Value Range

Type: Int

Value must be in the range [1..310]
LowMass must be less than HighMass

HighMass

Variable that controls the highest mass to sample when doing a mass sweep

Value Range

Type: Int

Value must be in the range [1..310]
HighMass must be greater than LowMass

SamplesPerAmu

Variable that controls the samples per amu to take when doing a mass sweep

Value Range

Type: Int

Value must be in the range [6..20]

ScanSpeed

Variable that controls the number of samples per second to take when doing a mass sweep

Value Range

Type: Int

Value must be one of [1000, 500, 288, 144, 72, 48, 24, 20, 12, 10, 6, 5, 3, 2, 1, 0.5, 0.2, 0.1]

AutoZero

Attempts to automatically zero the data

Value Range

Type: Int

Value must be one of [1(on), 0(off)]

AutoStream

Automatically initiates streams of sweep/trend data any time a new sweep/trend is initiated

Value Range

Type: Int

Value must be one of [1(on), 0(off)]

Filament

Turns the filament on/off

Value Range

Type: Int

Value must be one of [1(on), 0(off)]

MultiplierVolts

Sets the voltage to be applied to the electron multiplier in Volts

Value Range

Type: Int

Value must be in the range [0(off)..3000]

FilamentEmissionMa

Sets the filament emission current in milliamps(mA)

Value Range

Type: Int

Value must be in range [0.1 .. 4.0]

ElectronVolts

Sets the energy used in the ionization process

Value Range

Type: Int

Value must be in range [11..150]

Focus1Volts

Sets the focus voltage in Volts

Value Range

Type: Int

Value must be in the range [-150..0]

SamplesPerLine

Sets the samples per line for sweep data to be reported

Value Range

Type: Int

Value must be positive

Encoding

Sets the encoding scheme by which the sweep data is reported

Value Range

Type: Int

value must be one of 10, 16, 64

IsIdle

(read-only)

Reports if the RGA is in an idle state

Value Range

[0, 1]

LastSweep

(read-only)

Reports the last(most recent) sweep number saved in memory

Value Range

Type: Int

Value must be positive or 0

FirstSweep

(read-only)

Reports the first sweep number saved in memory

Value Range

Type: Int

Value must be positive or 0

GroundVolts

(read-only)

Returns ground voltage in Volts

Value Range

Type: float

ReferenceVolts

(read-only)

Returns reference voltage in Volts

Value Range

Type: float

PiraniTorr

(read-only)

Returns Pirani gauge pressure in Torr

Value Range

Type: float

PiraniVolts

(read-only)

Returns Pirani gauge voltage in Volts

Value Range

Type: float

PiraniOhms

(read-only)

Returns Pirani gauge resistance in Ohms

Value Range

Type: float

PiraniTempVolts

(read-only)

Returns Pirani gauge temperature in Volts

Value Range

Type: float

SupplyVolts

(read-only)

Returns power supply voltage in Volts

Value Range

Type: float

QuadrupoleDegC

(read-only)

Returns quadrupole temperature in Celsius

Value Range

Type: float

InteriorDegC

(read-only)

Returns interior temperature in Celsius

Value Range

Type: float

IonizerVolts

(read-only)

Returns ionizer voltage in Volts

Value Range

Type: float

IonizerAmps

(read-only)

Returns ionizer current in Amps

Value Range

Type: float

IonizerOhms

(read-only)

Returns ionizer resistance in Ohms

Value Range

Type: float

RfAmpVolts

(read-only)

Returns RF amp voltage in Volts

Value Range

Type: float

SourceGrid1Ma

(read-only)

Returns source 1 current in milliamps(mA)

Value Range

Type: float

SourceGrid2Ma

(read-only)

Returns source 2 current in milliamps(mA)

Value Range

Type: float

FilamentDacCoarse

(read-only)

Value Range

Type: float

FilamentDacFine

(read-only)

Value Range

Type: float

FilamentPowerPct

(read-only)

Value Range

Type: float

FbPlus

(read-only)

Value Range

Type: float

FbMinus

(read-only)

Value Range

Type: float

Focus1FB

(read-only)

Returns focus voltage in Volts

Value Range

Type: float

PiraniCorrVolts

(read-only)

Value Range

Type: float

RepellerVolts

(read-only)

Returns repeller voltage in Volts

Value Range

Type: float

PressureAmps

(read-only)

Returns total pressure in Amps

Value Range

Type: float

PressureTorr

(read-only)

Returns total pressure in Torr

Value Range

Type: float

FilamentStatus

(read-only)

Returns filament status

When FilamentStatus=4, the filament attempted to use too much power and tripped. If this state is reached, the filament must be turned off and back on using the Filament symbol in order for the filament to turn back on.

Value Range

Type: Int

[0, 1, 2, 3, 4]

FilamentStatus=0 – Filament is OFF
FilamentStatus=1 – Filament is waiting for rough vacuum
FilamentStatus=2 – Filament is waiting for high vacuum
FilamentStatus=3 – Filament is ON
FilamentStatus=4 – Filament Tripped

DegasMa

(read-only)

Returns the current applied to the filament in degas mode in milliamps(mA)

Value Range

Type: float

SerialNumber

(read-only)

Returns the serial number

Value Range

Type: Int

ModelNumber

(read-only)

Returns the model number

Value Range

Type: Int

PiraniZero

Calibration data for Pirani gauge

Value Range

Type: float

Pirani1ATM

Calibration data for Pirani gauge

Value Range

Type: float

SwSettleTicks

Calibration data for RGA

Value Range

Type: Int

RfSettleTicks

Calibration data for RGA

Value Range

Type: Int

TotalOffset

Calibration data for RGA

Value Range

Type: Int

PartialOffset
Calibration data for RGA

Value Range
Type: Int

LowCalMass
Calibration data for RGA

Value Range
Type: Int

LowCalResolution
Calibration data for RGA

Value Range
Type: Int

LowCalIonEnergy
Calibration data for RGA

Value Range
Type: float

LowCalPosition
Calibration data for RGA

Value Range
Type: float

HighCalMass
Calibration data for RGA

Value Range
Type: Int

HighCalResolution

Calibration data for RGA

Value Range

Type: int

HighCalIonEnergy

Calibration data for RGA

Value Range

Type: float

HighCalPosition

Calibration data for RGA

Value Range

Type: float

TotalCapPf

Calibration data for RGA

Value Range

Type: float

PartialCapPf

Calibration data for RGA

Value Range

Type: float

TotalSensitivity

Calibration data for RGA

Value Range

Type: float

PartialSensitivity

Calibration data for RGA

Value Range

Type: float

VersionMajor

Major version

Value Range

Type: Int

VersionMinor

Minor version

Value Range

Type: Int

BaudRate

Sets the baud rate used for serial communication

Value Range

Type: Int

Value must be one of 9600, 19200, 38400, 57600, 115200

DegasTimer

Sets the degas timer in seconds

For functionality, see Degas Mode

Value Range

Type: Int

Value must be in the range [0(off)..600]

Instruction Set

clearChannels

Function

clears all channels in the mass table

Format

clearChannels

Sets - all channels:

enabled = 0
amu = 0
dwell = 42 ms

Returns

Format

ok:all channels cleared

channel

Function

Sets and returns channel data

Arguments

channel number (optional) - Range: [0..11]; 12 total channels
amu (optional) - amu to scan
dwell (optional) - dwell time to scan in milliseconds
enabled (optional) - [0(disabled), 1(enabled)]

Format

channel:*channel number*(int):**amu**:*amu*(float):**dwell**:*dwell*(float):**enabled**:*enabled*(int)

Returns

Returns data from each channel, informing channel number, amu, dwell(samples/sec), enabled

Format

ok:**channel**:*channel number*(int):**amu**:*amu*(float):**dwell**:*dwell*(float):**enabled**:*enabled*(int)

trend

Function

Starts trend mode

Scans through each enabled channel at the specified parameters

Arguments

count (optional) - specifies a count of sweeps to do. If you do not specify a count, the CCU will trend continuously until stopped or until another sweep is commanded. The trend data stored can be retrieved using the FirstSweep and LastSweep variables. (default=infinity)

radius (optional) - sets how many samples to take around target, reports max(samples). Range: [0..3]

radius 0 means only the target amu.
radius 1 means one extra sample on each side (+/- 0.25 amu)
radius 2 means two extra samples on each side (+/- 0.25, +/- 0.50 amu)
radius 3 means three extra samples on each side
The default is radius 2, which makes 5 samples per target altogether.

size (optional) - how many datasets to take per trend pass (default is one).

Format

trend:**count**:*count*(int):**radius**:*radius*(int):**size**:*size*(int)

Returns

Format

inf:**FirstSweep**:*first sweep number*
inf:**LastSweep**:*last(or current) sweep number*

stream

sends sweep/trend data

Arguments

from (optional) – lowest amu to report (default: LowMass used to take sweep)
to (optional) – highest amu to report (default: HighMass used to take sweep)
sweep (optional) - sweep number to report (default: most recent sweep)

Format

stream:**sweep**:*sweep number*(int):**from**:*low mass*(int):**to**:*high mass*(int)

Returns

Format

On success:
BeginStream:LowMass:*from***:HighMass:***to***:SamplesPerAmu:***samples per amu***:sweep:***sweepnum*

On failure:
"error: sweep number *sweepnum* not present"

sweep

Starts sweep based on variables ScanSpeed, LowMass, HighMass, SamplesPerAmu.

Arguments

count (optional) - number of sweeps to perform before stopping (default = infinity)

Format

sweep:**count**:*count*(int)

Returns

Sweep information

Format

inf:**FirstSweep**:*first sweep number*
inf:**LastSweep**:*last(or current) sweep number*

stop

Function

Forces the RGA into an idle state

Stops sweeping

Stops streaming

Format

stop

Returns

N/A

symbols

Symbols are broken up into 4 categories:

outputs
calibration
controls
hardware

To see a complete list of the symbols and their functionalities, see the Symbols section.

Function

Returns all symbol values

Format

symbols

Returns

A list of all symbols

Format

ok:**symbol name**:*symbol value*

for all symbols

controls

The controls subset of all symbols

All control variables:

LowMass
HighMass
SamplesPerAmu
ScanSpeed
AutoZero
AutoStream
Filament
MultiplierVolts
FilamentEmissionMa
ElectronVolts
Focus1Volts
SamplesPerLine
Encoding

Format

controls

Returns

A list of all control variables

Format

ok:**symbol name**:*symbol value*

for all symbols in the control category

outputs

The outputs subset of all symbols

All output variables:

GroundVolts
ReferenceVolts
PiraniTorr
PiraniVolts
PiraniOhms
PiraniTempVolts
SupplyVolts
QuadrupoleDegC
InteriorDegC
IonizerVolts
IonizerAmps
IonizerOhms
RfAmpVolts
SourceGrid1Ma
SourceGrid2Ma
FilamentDacCoarse
FilamentDacFine
FilamentPowerPct
FbPlus
FbMinus
Focus1FB
PiraniCorrVolts
RepellerVolts
PressureAmps
PressureTorr
FilamentStatus
DegasMa
IsIdle
LastSweep
FirstSweep

Format

outputs

Returns

A list of all output variables

Format

ok:**symbol name**:*symbol value*

for all symbols in the outputs category

calibration

The calibration subset of all symbols

All calibration variables:

SerialNumber
ModelNumber
PiraniZero
Pirani1ATM
SwSettleTicks
RfSettleTicks
TotalOffset
PartialOffset
LowCalMass
LowCalResolution
LowCalIonEnergy
LowCalPosition
HighCalMass
HighCalResolution
HighCalIonEnergy
HighCalPosition
TotalCapPf
PartialCapPf
TotalSensitivity
PartialSensitivity
VersionMajor
VersionMinor

Format

calibration

Returns

A list of all calibration variables

Format

ok:**symbol name**:*symbol value*

for all symbols in the calibration category

hardware

The hardware subset of all symbols

All hardware variables:

BaudRate
DegasTimer

Format

hardware

Returns

A list of all hardware variables

Format

ok:**symbol name**:*symbol value*

for all symbols in the hardware category

get

Arguments

symbol name (required)

Format

get:*symbol name*

Returns

Requested symbol value

or

Error if symbol does not exist

Format

ok:**symbol name**:*symbol value*

or

error:symbol '*requested symbol*' unknown

set

Arguments

symbol name (required)
symbol value (required)

Format

set:*symbol name:symbol value*

Returns

On success

ok:**symbol name**:**symbol value**

On failure

(less than fields given in command)
error: too few fields in set command

OR (requested symbol name unknown)
error:symbol ‘*symbol name’* unknown

OR (error during assigning)
inf:**symbol name**:**symbol value**

OR (if requested symbol is read-only)
error: "*symbol name*" is read-only

How to Compile the Demo App

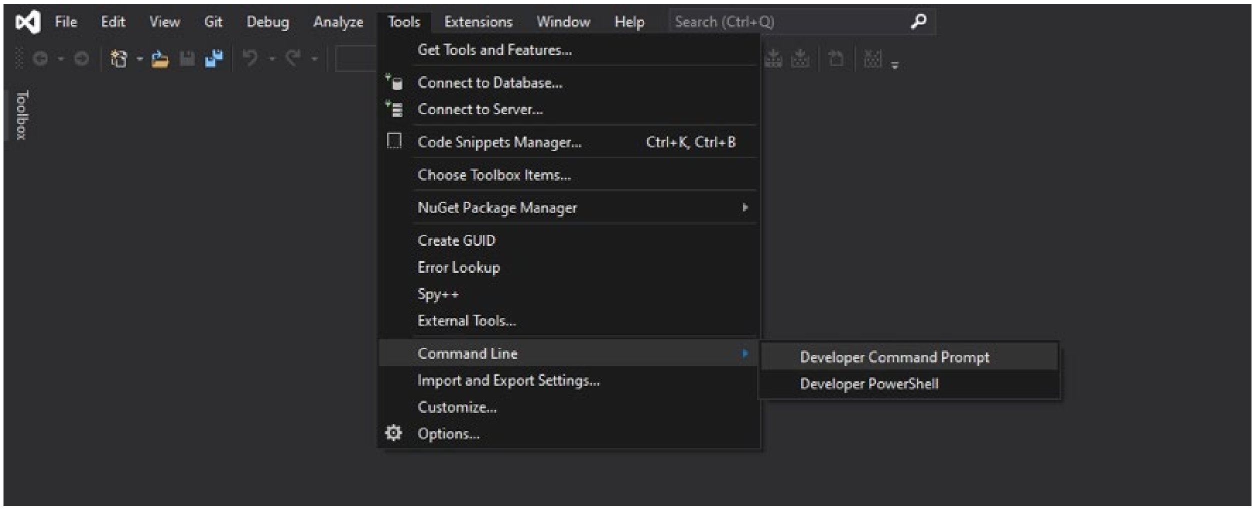
If you wish to make your own changes to our C# Demo App, you may do so by compiling the source code with your changes.

To aid with the compilation, we have provided a makefile that uses Microsoft NMAKE along with the source code in the rga directory.

To build the rga.exe program, invoke Microsoft nmake in the rga directory.

From Microsoft: *To use NMAKE, you must **run it in a Developer Command Prompt window**. A Developer Command Prompt window has the environment variables set for the tools, libraries, and include file paths required to build at the command line.*

As of Visual Studio 2019, the Developer Command Prompt can be found from:
Tools->Command Line->Developer Command Prompt



Invoking "nmake" builds the rga.exe and bootfirm.exe programs.

This makefile embeds the QpBox firmware, built separately, into the rga.exe executable. This firmware binary is expected to be in a file named qpbox.l2 in directory ..\qpbox.