PICARRO

19 AUG 2010 REVISION 2.0

Picarro Analyzer Programming Guide

Remote Command Interface

PICARRO, INC.
480 Oakmead Parkway
Sunnyvale, California
CA 94085
USA.

TABLE OF CONTENTS

TABLE OF TABLES

Modification History

Rev	Date	Author	Comment
0.1	Dec 1, 2005	Russell Warren	First Draft
0.2	Dec 5, 2005	Russell Warren	Added a few new commands
0.3	Dec 7, 2005	Russell Warren	Added measurement buffering commands and _Meas_GetConcEx; Changed –1 return code to "ERR"
0.4	Jan 30, 2005	Chris Rella, Russell Warren	Added walkthroughs, modified command descriptions to reflect implementation
0.5	Jun 15, 2006	Russell Warren	Added all EIF calls
0.6	May 18, 2007	Chris Rella	Added multiple concentrations to GET_CONC, etc. calls
0.7	July 26, 2007	Chris Rella	Added _MEAS_READVALUES call
0.8	May 22, 2008	Chris Rella	Updated format
0.9	May 27, 2008	Chris Rella	Added _VALVES_SEQ_START, _VALVES_SEQ_STOP, _VALVES_READSTATE, _VALVES_SETSTATE to command listing
1.0	July 28, 2008	Chris Rella	Added _PULSE_TRIGGER_ON, _PULSE_TRIGGER_OFF, _PULSE_GETBUFFERFIRST, _PULSE_GETBUFFER, and _PULSE_GETSTATUS to command listing
1.01	December 22, 2008	Chris Rella	Restored descriptions of _MEAS_GETBUFFERx command set
2.0	August 16, 2010	Alex Lee	Updated and implemented all the commands in Picarro G2000 platform using Python programming language.

Related Documents

Document Number	**Description**

Definitions

Acronym/Word	**Definition**
CRDS	Cavity Ring Down Spectroscopy
CRDI	Cavity Ring Down Instrument
CRD	Cavity Ring Down
FIFO	First In First Out

Document Purpose and Scope

This is a preliminary proposal for the external command interface on the CRDS instrumentation. This document describes the Picarro Analyzer's remote interface to use across TCP/IP or RS-232 and is viewable by selected external customers.

1 General Description

The instrument automatically warms up when power is applied. This process takes approximately 30 minutes. Measurement is possible when:

- Instrument is warmed up
- Gas is flowing through the instrument in a controlled and measurable state, with temperature and pressure stabilized

1.1 Communication hardware

Remote communication with the Picarro CRDS instrument can be done with either the provided RS-232 interface, or through via TCP/IP using the provided Ethernet port.

1.2 Communication Protocol

- RS-232 is **9600, N, 8, 1**
- Ethernet via TCP/IP. The analyzer IP address can be found using **"ipconfig"** command in Windows. The command interface port number is **51020**.

1.3 Comments on future compatibility of the preliminary command set

Underscore prefixes are used for the commands to allow easy compatibility in the future when we come out with our final interface protocol. Another way this could be done is to have a fixed command available to switch the mode of the command parser, but the preference is to stay with a recognizably different debug command set. When the final command interface is released we can still support these commands.

1.4 Command Syntax

- To execute a command, the format is:
  - <Command><space><P1><space><P2> <CR>
    - eg: _DO_SOMETHING 1 2<CR>
    - Angle brackets are not to be entered, they indicated special characters or parameters in this context.
    - <CR> is ascii character 13
    - Any linefeed (ascii 10, or <LF>) characters in the transmission are ignored
      - ie: it is okay to send a "typical" <CR><LF> at the end of a command because the <LF> will be ignored
    - P1 and P2 are function parameters (if there are any)
    - There can be more than two parameters, and if there are they should be separated by a single space.
    - If a function has no parameters, just the function name need be sent. eg: FUNCNAME<CR>
    - Case does not matter (eg: _Meas_GasConc is the same as _MEAS_GASCONC)

- All functions provide a return value
  - Return values are of the form:
    - <return value><CR>
    - <CR> is ASCII character 13
    - return values are always in ASCII format, not binary.
      - eg: a numeric return value of 10.2 would return as 5 bytes:
        - <49><48><46><50><13>
        - These are the ASCII codes for "1", "0", ".", "2", and the terminating <CR> code 13
    - Return values are specified in the function descriptions

1.5 Error Handling

If there is a problem with any executed command the return value will indicate this. For further detail consult the specific command documentation.

All errors have the following syntax: "ERR:####<TAB>*TIMESTAMP*<CR>" (no quotes), where the #### is a four digit error code which describes the error. Those communication or command related errors are returned immediately via the RS-232 interface; internal system errors are reported to the error buffer in the same format. Error codes are listed in Section 3 of this document.

To determine error states for the overall system and to download the contents of the Error buffer, use the `_Instr_GetStatus` command.

2 Command Reference

A summary of the available commands can be found in Table 1.

Table 1 – Summary of commands available through the programming interface

Command	***Description***
_Meas_GetConc	Retrieves the latest measured gas concentration from the instrument.
_Meas_GetConcEx	Retrieves the latest measured gas concentration from the instrument with a timestamp.
_Meas_GetBuffer	Retrieves several historical data measurements at one time from the instrument buffer.
_Meas_GetBufferFirst	Retrieves the earliest data point and time stamp from the measurement buffer.
_Meas_ClearBuffer	Clears the historical measurement buffer.
_Meas_Set_Tagalong_Data	Allows the user to integrate tagalong data from peripherals (e.g., GPS, weather station, etc) into the CRDS instrument log files and GUI.
_Meas_Get_Tagalong_Data	Retrieves the current value of the specified tagalong data
_Meas_Delete_Tagalong_Data	Removes the specified tagalong data from the CRDS instrument
_Meas_GetScanTime	Retrieves the approximate measurement interval.
_Instr_GetStatus	Gets the status of various instrument system components.
_EIF_AnalogOut_SetTracking	Puts an analog output into tracking mode.
_EIF_Analog_SetOutput	Sets an analog output to the specified output level.
_EIF_Analog_SetReference	Sets an analog output to a level corresponding to the specified measurement value.
_EIF_Analog_Configure	Configures the settings for an analog output line.
_EIF_Analog_GetInfo	Retrieves the information about an analog output line (including configuration).
_Valves_Seq_Start	Starts the Automatic Solenoid Valve Sequencer
_Valves_Seq_Stop	Stops the Automatic Solenoid Valve Sequencer
_Valves_Seq_Readstate	Reads the current state of the solenoid valves and sequencer
_Valves_Seq_Setstate	Manually sets the state of the solenoid valves
_Pulse_GetBufferFirst	Reports the oldest pulse analysis parameters in the buffer
_Pulse_GetBuffer	Dumps the entire pulse analysis buffer to the command interface
_Pulse_ClearBuffer	Empties the pulse analysis buffer without reporting its contents
_Pulse_GetStatus	Reports current trigger levels, ON/OFF state, etc.
_Flux_Mode_Switch	Switches a Flux analyzer to the desired scanning mode (Flux instruments only)

2.1 _Meas_GetConc

The **Meas_GetConc** function retrieves the latest measured concentration from the instrument.

Parameters
This function has no parameters.

Return Values
The return values are the latest gas concentration values.

eg: "558.692;1.859;1.630<CR>"

Each concentration value is reported with 3 numbers after the decimal. Multiple gas concentrations are separated by semicolon (;). There are no unit indications in the return value. For unit designations for each concentration, please refer to your user manual.

Possible Error Codes Specific to this Command

ERR:3001	Measurement system disabled

Comments
Unlike **_Meas_GetBuffer** or **_Meas_GenBufferFirst**, calling **_Meas_GenConc** will retrieve only the last data point *without* removing it from the FIFO buffer. Therefore repeatedly calling this command before the instrument taking any new measurement will result in identical return values. On the other hand, if the instrument takes measurements faster than the frequency of calling this command, some instrument measurements will be missing from the return values. If batch processing of instrument data is required, we do not recommend using the **_Meas_GenConc** command; instead, use **_Meas_GetBuffer** or **_Meas_GenBufferFirst** to download all data points available from the buffer.

2.2 _Meas_GetConcEx

The **Meas_GetConcEx** function retrieves the latest measured concentration from the instrument and the time at which the spectral scan corresponding to this measurement was completed.

Parameters
This function has no parameters.

Return Values
The return value are semicolon delimited data with the first element being the time stamp at which the spectral scan was completed, and the remaining elements are the reported gas concentrations.

eg: "10/08/17 23:25:22.086;571.019;1.860;1.623<CR>" is an example for 3 concentration measurements (571.019, 1.860, and 1.623) where the spectrum acquisition was completed on August 17, 2010 at 23:25 and 22.086 seconds.

The reason for specifying that the time is the time "at which the spectral scan was completed" is that there will be some (application dependent) additional processing time between the spectrum acquisition and the determination of a new gas concentration. Reporting the time at which actual physical spectrum was taken provides a better time synchronization between the measured value and what was actually happening in the sample cavity at what time. For most applications this processing lag is insignificant when compared to gas flow times.

The format of the date is always: "`YY/MM/DD HH:mm:ss.sss`", with HH being the hours on a 24 hr clock, and ss.sss being clock seconds to three decimal places.

Concentration is reported as in **_Meas_GetConc**.

Possible Error Codes Specific to this Command

`ERR:3001` Measurement system disabled

Comments

This is an extended version of the **_Meas_GetConc** function. The only difference is that the return value also contains the time stamp indicating when the optical measurement was completed.

As noted in the documentation for the **_Meas_GetConc** call, we do not recommend using this command for those instruments which operate in data batch mode. Use **_Meas_GetBuffer** or **_Meas_GetBufferFirst** instead.

2.3 _Meas_GetBuffer

The **_Meas_GetBuffer** function retrieves a set of measurements (and time stamps) from the measurement buffer.

Parameters

This function has no parameters.

Return Values

The return value is a string containing two separate sections:

1. The number of data records
2. A data section containing the measurement timestamp and the measured values

The first number to be returned is the total number of data records that are being returned, and it is followed by a carriage return. The data records are sent with the semicolon delimiting the time and each concentration value.

An example of the general form:

<N>;<CR>
<Time1>;<Meas1_Conc1>;<Meas1_Conc2>;..;<CR>
<Time2>;<Meas2_Conc1>;<Meas2_Conc2>;..;<CR>
..
..
<TimeN>;<MeasN_Conc1>;<MeasN_Conc2>;..;<CR>
<CR>

Where *N* is the number of data records. The semicolon character (ASCII value 59) is used to separate the starting sample count, each concentration value within a data record, and each data record.

An example of a return value containing two data records (each record has 3 concentration values) is as follows:

```
2;<CR>
10/08/18 12:02:23.480;0.285;0.021;0.002;<CR>
10/08/18 12:02:24.806;0.285;0.033;0.006;<CR>
<CR>
```

If no new measurements have been made since the buffer was cleared last, the return value is simply "0;<CR>".

Possible Error Codes Specific to this Command

ERR:3001	Measurement system disabled

Comments

The instrument has a FIFO buffer of 512 values. If more than 512 measurements are made without the buffer being cleared, the old data is lost. If un-cleared the buffer will always contain the most recent 512 values.
Calling this function clears the measurement FIFO buffer. The FIFO buffer can also be cleared with a call to **_Meas_ClearBuffer**.

Formatting of the time and measurement data is as is documented in the **_Meas_GetConc** and **_Meas_GetConcEx** functions.

As with **_Meas_GetConcEx**, the times reported by this call are the time at which the spectral measurement was completed, not the times at which they are delivered to the measurement buffer.

See also: **_Meas_ClearBuffer, _Meas_GetBufferFirst**

2.4 _Meas_GetBufferFirst

The **Meas_GetBufferFirst** function retrieves the earliest measured concentration from the instrument measurement buffer and the time at which the spectral scan was completed.

Parameters

This function has no parameters.

Return Values

The return value is a semicolon delimited data record with the first element being the time stamp at which the spectral scan was completed.

eg: "10/08/18 12:32:24.390;0.214;0.021;0.008;<CR>"

This example above shows how multiple concentrations are being reported with the same timestamp.

The reason for specifying that the time is the time "at which the spectral scan was completed" is that there will be some (application dependent) additional processing time between the spectrum acquisition and the determination of a new gas concentration. Reporting the time at which actual physical spectrum was taken provides a better time synchronization between the measured value and what was actually happening in the sample cavity at what time.

The format of the date is always: "YY/MM/DD HH:mm:ss.sss", with HH being the hours on a 24 hr clock, and ss.sss being clock seconds to three decimal places.

The concentration is always reported with 3 digits of precision. There are no unit indications in the return value.

Possible Error Codes Specific to this Command

ERR:3001	Measurement system disabled
ERR:3002	No measurements data exists

Comments

In many applications, measurements are delivered by the instrument into the measurement buffer in batches of several data points at a time. Calling this function repeatedly will remove data points one at a time from the buffer until the buffer is empty, starting with the earliest data point in the buffer.

See also: **_Meas_ClearBuffer, _Meas_GetBuffer**

2.5 _Meas_ClearBuffer

The **_Meas_ClearBuffer** function clears the measurement buffer FIFO.

Parameters

This function has no parameters.

Return Values

Returns "OK" for all situations.

Comments

This call can be used when starting a new measurement phase in order to avoid old data contaminating the retrieved data set.

See also: **_Meas_GetBuffer**, **_Meas_GetBufferFirst**

2.6 _Meas_Set_Tagalong_Data

The **_Meas_Set_Tagalong_Data** function allows the user to integrate tagalong data from peripherals (e.g., GPS, weather station, etc) into the CRDS instrument log files and GUI.

Parameters

Label:

The label of the tagalong data

Value:
The value of the tagalong data

Return Values
Returns “OK” on success.

Comments
If the specified tagalong data does not already exist, calling this command will create a new data column with the given initial value. The current active log file will be closed and a new log file will be generated to contain this new data column. If the tagalong data already exists, this command will simply update its value.

Example Usage
The following call adds a new data column called “Latitude” with initial value 41.03. If the “Latitude” column already exists, it will update its value to 41.03.

```
_Meas_Set_Tagalong_Data Latitude 41.03
```

See also: **_Meas_Get_Tagalong_Data**, **_Meas_Delete_Tagalong_Data**

2.7 _Meas_Get_Tagalong_Data

The **_Meas_Get_Tagalong_Data** function retrieves the current value of the specified tagalong data.

Parameters
Label:
The label of the tagalong data

Return Values
Returns “OK” on success.

Comments

See also: **_Meas_Set_Tagalong_Data**, **_Meas_Delete_Tagalong_Data**

2.8 _Meas_Delete_Tagalong_Data

The **_Meas_Delete_Tagalong_Data** function removes the specified tagalong data from the CRDS instrument log files and GUI.

Parameters
Label:
The label of the tagalong data

Return Values
Returns “OK” on success.

Comments

See also: **_Meas_Set_Tagalong_Data**, **_Meas_Get_Tagalong_Data**

2.9 _Meas_GetScanTime

The **Meas_GetScanTime** function returns the approximate time it takes for the measurement system to make a measurement of a gas concentration.

Parameters
This function has no parameters.

Return Values
The return value is the approximate time (in seconds) between concentration points.

Comments
This is an approximate time.

2.10 _Instr_GetStatus

The **_Instr_GetStatus** function returns the status of the instrument status register, which indicates the status of various system elements with one call.

Parameters
This function has no parameters.

Return Values
Returns an integer number representing the contents of the 16 bit instrument status register.

Comments
Each bit of the register is described in Table 2.

Table 2 – The Instrument Status Register

Bit Number (0 = LSB, 15 = MSB)	***Decimal Value***	***Mnemonic***	***Description of set condition***
15	32768	<reserved>	This bit currently has no meaning and should be ignored.
14	16384	System Error	0 = The instrument is not currently in an error state 1 = A system error is present. Use _Instr_GetError for more information.
13	8192	Warming up	0 = The instrument has successfully started up 1 = The instrument is currently warming up from power-off or restart
9	512	Warm box temp locked	0 = The warm box temperature is not stabilized within acceptable bounds 1 = The warm box temperature is within acceptable bounds for measurements
8	256	Cavity temp locked	0 = The cavity temperature is not stabilized within acceptable bounds 1 = The cavity temperature is within acceptable bounds for measurements
7	128	Pressure locked	0 = The gas sample pressure is not stabilized within acceptable bounds 1 = The gas sample pressure is within acceptable bounds for measurements
6	64	Gas Flowing	0 = Valves are closed and no gas is flowing 1 = Valves are open (pressure not necessarily stable)
2	4	Error in buffer	0 = The error queue is empty 1 = There is at least one value in the error queue
1	2	Meas Active	0 = The measurement system is currently inactive 1 = The measurement system is currently active
0	1	Ready	0 = The instrument currently cannot make a gas measurement 1 = The instrument is currently capable of measuring the sample gas

Usually when the instrument is under operational condition and taking measurements, the return value should be **963** (= Bit 0 (ready) AND Bit 1 (measurement active) AND Bit 6 (gas flowing) AND Bit 7 (pressure locked) AND Bit 8 (cavity temperature locked) AND Bit 9 (warm box temperature locked)).

Additional information on each bit follows:

Bit 0 – Ready: Gas measurements are possible as:
- The instrument is warmed up
- The conditions in the sample cavity are acceptable (pressure and temperature controlled within range)
- The instrument is not busy doing something else.

If bits 0 and 1 are both set (return value = 3) it means the instrument is currently measuring.

Bit 1 – Measurement inactive/active: is set LOW when the measurement system is inactive, HIGH when measurements are in progress.

Bit 2 – Error in buffer: is set whenever a system error is present in the error buffer. This bit is not cleared until the buffer has been emptied. In general, errors that occur exclusively in the command interface (error codes 1000-1999) do not result in an error being logged in the error queue.

Bit 6 – Gas flowing: is set LOW unless the inlet and outlet valve are both open.

Bit 7 – Pressure locked: is set LOW when the pressure is outside of acceptable operating range, OR gas is not flowing. If the pressure is unable to lock for an extended period when it should, this can be the result of an over or under pressure at the sample input, or a loss of vacuum.

Bit 8 – Cavity temperature locked: is set LOW when the cavity temperature is outside of acceptable operating range.

Bit 9 – Warm box temperature locked: is set LOW when the warm box temperature is outside of acceptable operating range.

Bit 13 – Starting up: is set HIGH immediately after the instrument powers up. This bit clears when the instrument has completed the warmup time (instrument is temperature stabilized) and should then never be set again until the instrument is restarted.

Bit 14 – System Error: If set, _Instr_GetError can be called to determine what error occurred. This bit will remain HIGH until error condition no longer exists. All error conditions that cause this bit to be set will also generate an error entry in the error log (and set bit 2 high). However, it is possible for this bit (bit 14) to be high when bit 2 is LOW. This can happen when a persistent error condition exists, the error log is read (clearing bit 2), but the error condition still exists. This bit is not set for errors generated at the RS-232 interface (error codes 1000-1999)

2.11 _EIF_AnalogOut_SetTracking

The **_EIF_AnalogOut_SetTracking** function puts a specified analog output line into tracking mode.

Parameters
Channel:
The analog output channel to deal with. Note that the output channels are numbered from 0.

Return Values
Returns “OK” on success.

Possible Error Codes Specific to this Command

ERR:5001	Invalid channel specified

Comments
If the specified output is already in Tracking mode, this function has no effect.

2.12 _EIF_ AnalogOut_SetOutput

The **_EIF_AnalogOut_SetOutput** function sets the analog output to the specified output level.

Parameters
Channel:
The analog output channel to deal with. Note that the output channels are numbered from 0.

OutputLevel:
The output level to set. Unit is Volt.

Return Values
Returns "OK" on success.

Possible Error Codes Specific to this Command

ERR:5001	Invalid channel specified

Comments
If not already in Manual mode, executing this command forces the output into Manual mode. To return to tracking mode a call must be made to **_EIF_AnalogOut_SetTracking.**

Example Usage
The following call would set Analog Voltage Output #2 to 3.753 Volts:

```
_EIF_ANALOGOUT_SETOUTPUT 2 3.753
```

2.13 _EIF_ AnalogOut_SetReference

The **_EIF_AnalogOut_SetReference** function sets the analog output to a value that corresponds to a specified measurement value.

Parameters
Channel:
The analog output channel to deal with. Note that the output channels are numbered from 0.

MeasurementLevel:
The measurement value that should be output to the specified analog output. Unit is the same as the concentration unit (ppmv, etc).

Return Values
Returns "OK" on success.

Possible Error Codes Specific to this Command

ERR:5001	Invalid channel specified

Comments

This function differs from **_EIF_AnalogOut_SetOutput** in that an actual gas measurement value is specified instead of a direct analog output level. In this way, any reference value can be set on the analog output in order to calibrate any external instrumentation hooked up to the analog output(s).

The analog output value set emulates the value that would occur if the analog output were in Tracking mode and the instrument measured a value equal to the specified *MeasurementLevel* value. This value is determined by the following equation:

Analog output value = <Cal_Slope>**MeasurementLevel* + <Cal_Offset>

where <Cal_Slope> and <Cal_Offset> are the calibration parameters associated with the indicated *Channel*.

If the resulting output level is outside of the limitations configured for the analog output, the return value will be clipped.

If not already in Manual mode, executing this command forces the output into Manual mode. To return to tracking mode a call must be made to **_EIF_AnalogOut_SetTracking**.

Example Usage
The following call would set Analog Voltage Output #2 to the voltage corresponding to 0 ppmv:

```
_EIF_ANALOGOUT_SETREFERENCE 2 0.0
```

In this case, the output could then be used to get a precise calibration of any offset voltages at zero ppmv between the instrument and the connected voltage measurement device.

2.14 _EIF_ AnalogOut_Configure

The **_EIF_AnalogOut_Configure** function is used to configure the settings for the specified analog output.

Parameters
Channel:
The analog output channel to deal with. Note that the output channels are numbered from 0.

CalSlope:
The calibration slope (V/ppmv) to use when the output is in Tracking mode, or when **_EIF_AnalogOut_SetReference** is called.

CalOffset:
The calibration offset (V) to use when the output is in Tracking mode, or when **_EIF_AnalogOut_SetReference** is called.

MinOutput:
The absolute minimum level to allow on the specified analog output. The default value is 0 Volt.

MaxOutput:
The absolute maximum level to allow on the specified analog output. The default value is 10 Volts.

BootMode:
What mode the output should be in on power-up (0=Manual, 1=Tracking).

BootValue:
The value that should be shown on the output on power up. Unit is Volt.

InvalidLevel:
The level that the output should go to when the measurement is invalid. Unit is Volt.

Return Values
Returns "OK" on success.

Possible Error Codes Specific to this Command

ERR:5001	Invalid channel specified

Comments
When the analog output is in Tracking mode any measurements that would result in an output outside of the range specified by *MinLevel* and *MaxLevel* will be clipped to one of the two.

If the requested *MinOutput* or *MaxOutput* are out of range of the physical limitations of the analog output, their values will be modified based on the physical limitations.

Example Usage
The following call configures analog output voltage #2 for a maximum possible output range of 0-5V, with 0 ppmv being output as 0.1 V, and 1 ppmv being output as 4.1V. It starts from Tracking mode with initial output = 0 V. The invalid level is also specified as 0 V.

```
_EIF_ANALOGOUT_CONFIGURE 2 4 0.1 0 5 1 0 0
```

2.15 _EIF_AnalogOut_GetInfo

The **_EIF_AnalogOut_GetInfo** function retrieves configuration and status information for the specified analog output.

Parameters
Channel:
The analog output channel to deal with. Note that the output channels are numbered from 0.

Return Values
The return value is a semicolon (; = ASCII value 59) delimited set of the configuration parameters for the specified output.

The parameters that are returned, in order, are:

CurrentState – the current state of the output (0=Manual, 1=Tracking)
MeasSource – the measurement source and concentration associated with the output. The measurement source and concentration are separated by a comma (,).
CalSlope – the calibration slope for measurement to output conversion
CalOffset – the calibration offset for measurement to output conversion
MinOutput – the minimum output level that will be output
MaxOutput – the maximum output level that will be output
BootMode – the mode at power-up (0=Manual, 1=Tracking)
BootValue – the output value at power-up
InvalidValue – the level that will be output when the measurement is invalid
CurrentValue – the level that is on the output at the time of this call (in mV or uA)

For description of some of these values, see the configuration description in the previous section.

An example response string that could result from a request for information on a voltage output is:

```
1;analyze_CFADS,ch4_conc;2.5;0.1;0;10;1;0;0;3.89876;<CR>
```

in which case, the specified voltage output:
- is currently in Tracking mode (*CurrentState* = 1)
- is configured to represent CH4 measurement from CFADS analyzer (*MeasSource* = analyze_CFADS,ch4_conc)
- is operating with a cal slope of 2.5 V/ppmv (*CalSlope* = 2.5)
- will output 0 ppmv as 0.1 V (*CalOffset* = 0.1)
- will never output below 0 V (*MinOutput* = 0)
- will never output above 10 V (*MaxOutput* = 10)
- will start from Tracking mode at power-up (*BootMode* = 1)
- will output 0 V at power-up (*BootValue* = 0)
- will output 0 V when there is no valid measurement (*InvalidValue* = 0)
- is currently outputting 3.89876 V, corresponding to a CH4 concentration of 1.5195 ppmv (*CurrentValue* = 3.89876)

2.16 _Valves_Seq_Start

The **_Valves_Seq_Start** function enables the automatic solenoid valve sequencer on the instrument.

Parameters
This function has no parameters.

Return Values
Returns “OK”.

Comments
The automatic solenoid valve sequencer is configured from the GUI --- please see the user manual for more information.

2.17 _Valves_Seq_Stop

The **_Valves_Seq_Stop** function disables the automatic solenoid valve controller on the instrument. The valves are left in their current state.

Parameters
This function has no parameters.

Return Values
Returns "OK".

2.18 _Valves_Seq_Readstate

The **_Valves_Readstate** reads the current state of the automatic solenoid valve sequencer (ON or OFF), as well as the specific valve configuration.

Parameters
This function has no parameters.

Return Values
Returns the sequencer state (OFF or ON), followed by a semicolon, followed by the binary value corresponding to the valve state. For example, a response code of ON;8 indicates the sequencer is on, and the all the valves are OFF except the valve three, which is in the ON state.

Comments
The automatic solenoid valve sequencer is configured from the GUI --- please see the user manual for more information.

2.19 _Valves_Seq_Setstate

The **_Valves_Setstate** set the solenoids to a manual configuration. Turns off the valve sequencer if it is already on.

Parameters
This function has a single parameter, which is a binary value corresponding to the desired state of the solenoid valves. Not all valve states are permitted. Consult the user manual or contact the factory for more information. For example _Valves_setstate 6 sets all valves to OFF except valves 1 and 2.

Return Values
Returns the sequencer state (OFF or ON), followed by a semicolon, followed by the binary value corresponding to the valve state. For example, a response code of ON;8 indicates the sequencer is on, and the all the valves are OFF except the valve three, which is in the ON state.

Comments

The automatic solenoid valve sequencer is configured from the GUI --- please see the user manual for more information.

2.20 _Pulse_GetBuffer

The **_Pulse_GetBuffer** function retrieves all pulse analysis results (and time stamps) from the pulse analysis buffer.

Parameters
This function has no parameters.

Return Values
The return value is a string containing two separate sections:

1. The number of data records
2. A data section containing the timestamp and the corresponding pulse analysis values

The first number to be returned is the total number of data records that are being returned, and it is followed by a carriage return. The pulse analysis outputs are sent with the semicolon delimiting the time and each returned value. Each concentration defined in the pulse analyzer will have three values returned – *mean*, *standard deviation*, and *slope*. A semicolon is used to separate each set of a concentration.

An example of the general form:

```
<N>;<CR>
<Time1>;<Conc1_Mean>;<Conc1_Std>;<Conc1_Slope>;<Conc2_Mean>;..;<CR>
<Time2>;<Conc1_Mean>;<Conc1_Std>;<Conc1_Slope>;<Conc2_Mean>;..;<CR>
..
..
<TimeN>;<Conc1_Mean>;<Conc1_Std>;<Conc1_Slope>;<Conc2_Mean>;..;<CR>
<CR>
```

Where *N* is the number of data records. The semicolon character (ASCII value 59) is used to separate the starting sample count, each pulse analysis value within a data record, and each data record.

An example of a return value containing 3 pulse analysis records (each record has 3 concentrations) is as follows:

```
3;<CR>
10/08/19 12:58:53.856;19272.113;75.147;1.000;-13.713;0.162;-0.001;-
106.438;0.532;0.009;<CR>
10/08/19 13:02:48.392;19693.978;80.396;0.802;-6.948;0.307;-0.010;-
6.547;1.214;0.041;<CR>
10/08/19 13:06:42.145;19271.733;116.508;1.547;-6.781;0.222;-0.007;-
2.237;0.758;0.015;<CR>
<CR>
```

This example was obtained from an isotopic water instrument, where the first 3 values after the time stamp represent the mean, standard deviation, and slope of H2O concentration, and the following 6 values represent the statistics for Delta 18_15 and Delta D_H respectively.

Possible Error Codes Specific to this Command

ERR:6001	No pulse analyzer data exists
ERR:6003	Pulse analyzer is not running

If no new pulse analysis data have been made since the buffer was cleared last, "No pulse analyzer data exists" error will be reported when using this command. If the pulse analyzer is not set up to run on the instrument, calling this command will result in "Pulse analyzer is not running" error.

Comments
The pulse analyzer has a FIFO buffer of 512 values. If more than 512 data records are made without the buffer being cleared, the old data is lost. If un-cleared the buffer will always contain the most recent 512 values.
Calling this function clears the pulse analysis FIFO buffer. The FIFO buffer can also be cleared with a call to **_Pulse_ClearBuffer**.

Formatting of the time and measurement data is as is documented in the **_Meas_GetConc** and **_Meas_GetConcEx** functions.

2.21 _Pulse_GetBufferFirst

The **_Pulse_GetBufferFirst** function retrieves the earliest pulse analysis values and time stamps from the instrument pulse analyzer.

Parameters
This function has no parameters.

Return Values
The return value is a semicolon delimited data list with the first element being the time stamp, and the rest elements are the reported pulse analysis values.

eg: "10/08/19 15:45:54.076;19635.708;56.578;0.526;-21.142;0.179;-0.003;-149.819;0.576;-0.012;"

The return value has the identical format as the output of command **_Pulse_GetBuffer**, except that it doesn't return the total number of data records in the first line (the number of data records is always 1).

Possible Error Codes Specific to this Command

ERR:6001	No pulse analyzer data exists
ERR:6003	Pulse analyzer is not running

If no new pulse analysis data have been made since the buffer was cleared last, “No pulse analyzer data exists” error will be reported when using this command. If the pulse analyzer is not set up to run on the instrument, calling this command will result in “Pulse analyzer is not running” error.

2.22 _Pulse_ ClearBuffer

The **_Pulse_ClearBuffer** function clears the pulse analysis buffer FIFO.

Parameters
This function has no parameters.

Return Values
Returns “OK” on success.

Comments
This call can be used when starting a new pulse analysis in order to avoid old data contaminating the retrieved data set.

2.23 _Pulse_ GetStatus

The **_Pulse_GetStatus** function returns status of the pulse analysis trigger state.

Parameters
This function has no parameters.

Return Values
0 – Waiting
1 – Armed
2 - Triggered

2.24 _Flux_Mode_Switch

The **_Flux_Mode_Switch** command switches a Flux instrument to run in the desired mode.

Parameters
Mode:
The desired flux mode, which can be chosen from:
CO2_H2O
H2O_CH4
CO2_CH4

Return Values
Returns “OK” on success.

Comments
This command is only used in Picarro Flux instruments.

3 Command script walkthroughs

In this section we provide command script walkthroughs to aid in the development of the automation software necessary to communicate with the CRDS instrument via the command interface.

The main things to remember when sending an RS232 command to the instrument are:

- All commands sent to the instrument (queries) must have a terminating <CR>
  - <CR> has an ASCII value of 13
- The instrument will *always* provide a response to the command
  - If it doesn't, there is a communication problem
- All responses from the instrument are ASCII strings
  - Numeric responses (like gas concentrations) come back as ASCII strings and need to be converted from strings to numbers
- All responses from the instrument will always have a terminating <CR>
- If there was a problem with the command, the response will *always* begin with "ERR:" as the first 4 characters, followed by a code indicating the type of error.
  - These command errors do not indicate errors with the instrument and will not appear in the instrument error queue.
- The CPU that is servicing command requests is the same CPU that is performing the measurement
  - ie: Do not poll the RS232 interface as fast as possible
  - It is best to wait for a response from the instrument prior to submitting the next query.
- Command syntax is not case sensitive
  - Calling `_INSTR_GETSTATUS` is the same as calling `_Instr_GetStatus`

Each of the following examples has a sample code fragment that performs the task described in the example. The language of the code is irrelevant and the code is incomplete – it is provided purely as another method of explaining the logic in the provided examples.

For reference, the language used in the example programs is Python – this was chosen for the examples due to the readability of this language. Any part of a line following the # character is a comment, and indentation of lines of code is important for defining conditional blocks, loop blocks, and function definitions.

For the sake of simplicity and brevity, none of the code examples have any proper error handling (One slight exception to this is the error identification in the ExecCmd function). Proper implementation of interfacing code should handle errors appropriately.

3.1 Example 1 – Command Query and Response Format

All communication with the instrument through the RS-232 interface should follow the following general sequence:

1. Write the command to the instrument
    a. Send the command string with a terminating <CR> (eg: _INSTR_GETSTATUS<CR>)
2. Read the response from the instrument
    a. Read the RS232 buffer one character at a time until a <CR> is encountered
3. Deal with the response
    a. Check for errors (look for "ERR:" in the first 4 characters)
    b. Use the response as appropriate

3.1.1 Sample Code

The following code fragment shows an example of the execution above.

```
def ExecCmd(Command):
  #send the command with the appropriate <CR> terminator...
  RS232.write(Command + chr(13))
  #and collect the response...
  buf = ""
  while 1:
    c = RS232.Read(1)   # reading one byte at a time
    if c == chr(13):    # until a <CR> is read
      break             # and then stop the reading loop
    else:
      buf = buf + c     # build the response (w/o <CR>)
  #Check if there is an error (first 4 chars = "ERR")...
  if buf[:4] == "ERR:":
    #There was an error - raise an exception with the message...
    raise Exception(buf)
  else:
    #No error - return the response string (w/o <CR>)...
    return buf
```

3.2 Example 2 – Collect data using _Meas_GetBufferFirst

One way to get the instrument data with `_Meas_GetBufferFirst` is to repeatedly make this call and ignore all return codes of "`ERR:3002`" (no data) that happen. This method has the disadvantage of not getting all of the other system information that is available in the status register that might be useful for something such as updating a Graphical User Interface.

3.2.1 Sample Code

```
while collectData == True:
    try:
        ret = ExecCmd("_MEAS_GETBUFFERFIRST")
        # parse the data string with semicolon
        ret = ret.split(";")
        sampleTime  = ret[0]   # time is always the first element
        print "Time = ", sampleTime
        for value in ret[1:]:   # conc always starts from the second element
            print "Value = ", value
    except:
        time.sleep(1.0)
# end of collectData loop
```

4 Error Codes

All possible error codes are listed below with a description of each.

1000	Communication failed
1001	Processing previous command
1002	Command not recognized
1003	Parameters invalid
1004	Command execution failed
3001	Measurement system disabled
3002	No measurements data exists
5001	Invalid channel specified (for Electrical Interface)
6001	No pulse analyzer data exists.
6002	Pulse analyzer is not running.