NAME

dcload

FILE

/mnt/c/users/blake/desktop/PyLib85xx/dcload.py

DESCRIPTION

Open Source Initiative OSI - The MIT License:Licensing

Tue, 2006-10-31 04:56 - nelson

The MIT License

Copyright (c) 2009 BK Precision

Permission is hereby granted, free of charge, to any person obtaining a copy

of this software and associated documentation files (the "Software"), to deal

in the Software without restriction, including without limitation the rights

to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

copies of the Software, and to permit persons to whom the Software is

furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in

all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN

THE SOFTWARE.

This python module provides a functional interface to a B&K DC load

through the DCLoad object. This object can also be used as a COM

server by running this module as a script to register it. All the

DCLoad object methods return strings. All units into and out of the

DCLoad object's methods are in SI units.

See the documentation file that came with this script.

$RCSfile: dcload.py $

$Revision: 1.0 $

$Date: 2008/05/17 15:57:15 $

$Author: Don Peterson $

CLASSES

InstrumentInterface

DCLoad

exceptions.Exception(exceptions.BaseException)

InstrumentException

class DCLoad(InstrumentInterface)

| Methods defined here:

|

| DisableLocalControl(self)

| Disable local control of the load

|

| EnableLocalControl(self)

| Enable local control (i.e., key presses work) of the load

|

| GetBatteryTestVoltage(self)

| Gets the battery test voltage

|

| GetCCCurrent(self)

| Gets the constant current mode's current level

|

| GetCRResistance(self)

| Gets the constant resistance mode's resistance level

|

| GetCVVoltage(self)

| Gets the constant voltage mode's voltage level

|

| GetCWPower(self)

| Gets the constant power mode's power level

|

| GetFunction(self)

| Get the function (type of operation) of the load

|

| GetInputValues(self)

| Returns voltage in V, current in A, and power in W, op\_state byte,

| and demand\_state byte.

|

| GetLoadOnTimer(self)

| Gets the time in seconds that the load will be on

|

| GetLoadOnTimerState(self)

| Gets the load on timer state

|

| GetMaxCurrent(self)

| Returns the maximum current the load will sink

|

| GetMaxPower(self)

| Gets the maximum power the load will allow

|

| GetMaxVoltage(self)

| Gets the maximum voltage the load will allow

|

| GetMode(self)

| Gets the mode (constant current, constant voltage, etc.

|

| GetProductInformation(self)

| Returns model number, serial number, and firmware version

|

| GetRemoteSense(self)

| Get the state of remote sensing

|

| GetTransient(self, mode)

| Gets the transient mode settings

|

| GetTriggerSource(self)

| Get how the instrument will be triggered

|

| Initialize(self, com\_port, baudrate, address=0)

| Initialize the base class

|

| RecallSettings(self, register=0)

| Restore instrument settings from a register

|

| SaveSettings(self, register=0)

| Save instrument settings to a register

|

| SetBatteryTestVoltage(self, min\_voltage)

| Sets the battery test voltage

|

| SetCCCurrent(self, current)

| Sets the constant current mode's current level

|

| SetCRResistance(self, resistance)

| Sets the constant resistance mode's resistance level

|

| SetCVVoltage(self, voltage)

| Sets the constant voltage mode's voltage level

|

| SetCWPower(self, power)

| Sets the constant power mode's power level

|

| SetCommunicationAddress(self, address=0)

| Sets the communication address. Note: this feature is

| not currently supported. The communication address should always

| be set to 0.

|

| SetFunction(self, function='fixed')

| Set the function (type of operation) of the load.

| function is one of "fixed", "short", "transient", or "battery".

| Note "list" is intentionally left out for now.

|

| SetLoadOnTimer(self, time\_in\_s)

| Sets the time in seconds that the load will be on

|

| SetLoadOnTimerState(self, enabled=0)

| Enables or disables the load on timer state

|

| SetLocalControl(self)

| Sets the load to local control

|

| SetMaxCurrent(self, current)

| Sets the maximum current the load will sink

|

| SetMaxPower(self, power)

| Sets the maximum power the load will allow

|

| SetMaxVoltage(self, voltage)

| Sets the maximum voltage the load will allow

|

| SetMode(self, mode)

| Sets the mode (constant current, constant voltage, etc.

|

| SetRemoteControl(self)

| Sets the load to remote control

|

| SetRemoteSense(self, enabled=0)

| Enable or disable remote sensing

|

| SetTransient(self, mode, A, A\_time\_s, B, B\_time\_s, operation='continuous')

| Sets up the transient operation mode. mode is one of

| "CC", "CV", "CW", or "CR".

|

| SetTriggerSource(self, source='immediate')

| Set how the instrument will be triggered.

| "immediate" means triggered from the front panel.

| "external" means triggered by a TTL signal on the rear panel.

| "bus" means a software trigger (see TriggerLoad()).

|

| TimeNow(self)

| Returns a string containing the current time

|

| TriggerLoad(self)

| Provide a software trigger. This is only of use when the trigger

| mode is set to "bus".

|

| TurnLoadOff(self)

| Turns the load off

|

| TurnLoadOn(self)

| Turns the load on

|

| ----------------------------------------------------------------------

| Methods inherited from InstrumentInterface:

|

| CalculateChecksum(self, cmd)

| Return the sum of the bytes in cmd modulo 256.

|

| CodeInteger(self, value, num\_bytes=4)

| Construct a little endian string for the indicated value. Two

| and 4 byte integers are the only ones allowed.

|

| CommandProperlyFormed(self, cmd)

| Return 1 if a command is properly formed; otherwise, return 0.

|

| DecodeInteger(self, str)

| Construct an integer from the little endian string. 1, 2, and 4 byte

| strings are the only ones allowed.

|

| DumpCommand(self, bytes)

| Print out the contents of a 26 byte command. Example:

| aa .. 20 01 .. .. .. .. .. ..

| .. .. .. .. .. .. .. .. .. ..

| .. .. .. .. .. cb

|

| GetCommand(self, command, value, num\_bytes=4)

| Construct the command with an integer value of 0, 1, 2, or

| 4 bytes.

|

| GetData(self, data, num\_bytes=4)

| Extract the little endian integer from the data and return it.

|

| GetIntegerFromLoad(self, cmd\_byte, msg, num\_bytes=4)

| Construct a command from the byte in cmd\_byte, send it, get

| the response, then decode the response into an integer with the

| number of bytes in num\_bytes. msg is the debugging string for

| the printout. Return the integer.

|

| GetReserved(self, num\_used)

| Construct a string of nul characters of such length to pad a

| command to one less than the packet size (leaves room for the

| checksum byte.

|

| PrintCommandAndResponse(self, cmd, response, cmd\_name)

| Print the command and its response if debugging is on.

|

| Reserved(self, num\_used)

|

| ResponseStatus(self, response)

| Return a message string about what the response meant. The

| empty string means the response was OK.

|

| SendCommand(self, command)

| Sends the command to the serial stream and returns the 26 byte

| response.

|

| SendIntegerToLoad(self, byte, value, msg, num\_bytes=4)

| Send the indicated command along with value encoded as an integer

| of the specified size. Return the instrument's response status.

|

| StartCommand(self, byte)

|

| ----------------------------------------------------------------------

| Data and other attributes inherited from InstrumentInterface:

|

| convert\_current = 10000.0

|

| convert\_power = 1000.0

|

| convert\_resistance = 1000.0

|

| convert\_voltage = 1000.0

|

| debug = 0

|

| highest\_register = 25

|

| length\_packet = 26

|

| lowest\_register = 1

|

| modes = {'cc': 0, 'cr': 3, 'cv': 1, 'cw': 2}

|

| to\_ms = 1000

class InstrumentException(exceptions.Exception)

| Method resolution order:

| InstrumentException

| exceptions.Exception

| exceptions.BaseException

| \_\_builtin\_\_.object

|

| Data descriptors defined here:

|

| \_\_weakref\_\_

| list of weak references to the object (if defined)

|

| ----------------------------------------------------------------------

| Methods inherited from exceptions.Exception:

|

| \_\_init\_\_(...)

| x.\_\_init\_\_(...) initializes x; see help(type(x)) for signature

|

| ----------------------------------------------------------------------

| Data and other attributes inherited from exceptions.Exception:

|

| \_\_new\_\_ = <built-in method \_\_new\_\_ of type object>

| T.\_\_new\_\_(S, ...) -> a new object with type S, a subtype of T

|

| ----------------------------------------------------------------------

| Methods inherited from exceptions.BaseException:

|

| \_\_delattr\_\_(...)

| x.\_\_delattr\_\_('name') <==> del x.name

|

| \_\_getattribute\_\_(...)

| x.\_\_getattribute\_\_('name') <==> x.name

|

| \_\_getitem\_\_(...)

| x.\_\_getitem\_\_(y) <==> x[y]

|

| \_\_getslice\_\_(...)

| x.\_\_getslice\_\_(i, j) <==> x[i:j]

|

| Use of negative indices is not supported.

|

| \_\_reduce\_\_(...)

|

| \_\_repr\_\_(...)

| x.\_\_repr\_\_() <==> repr(x)

|

| \_\_setattr\_\_(...)

| x.\_\_setattr\_\_('name', value) <==> x.name = value

|

| \_\_setstate\_\_(...)

|

| \_\_str\_\_(...)

| x.\_\_str\_\_() <==> str(x)

|

| \_\_unicode\_\_(...)

|

| ----------------------------------------------------------------------

| Data descriptors inherited from exceptions.BaseException:

|

| \_\_dict\_\_

|

| args

|

| message

class InstrumentInterface

| Provides the interface to a 26 byte instrument along with utility

| functions.

|

| Methods defined here:

|

| CalculateChecksum(self, cmd)

| Return the sum of the bytes in cmd modulo 256.

|

| CodeInteger(self, value, num\_bytes=4)

| Construct a little endian string for the indicated value. Two

| and 4 byte integers are the only ones allowed.

|

| CommandProperlyFormed(self, cmd)

| Return 1 if a command is properly formed; otherwise, return 0.

|

| DecodeInteger(self, str)

| Construct an integer from the little endian string. 1, 2, and 4 byte

| strings are the only ones allowed.

|

| DumpCommand(self, bytes)

| Print out the contents of a 26 byte command. Example:

| aa .. 20 01 .. .. .. .. .. ..

| .. .. .. .. .. .. .. .. .. ..

| .. .. .. .. .. cb

|

| GetCommand(self, command, value, num\_bytes=4)

| Construct the command with an integer value of 0, 1, 2, or

| 4 bytes.

|

| GetData(self, data, num\_bytes=4)

| Extract the little endian integer from the data and return it.

|

| GetIntegerFromLoad(self, cmd\_byte, msg, num\_bytes=4)

| Construct a command from the byte in cmd\_byte, send it, get

| the response, then decode the response into an integer with the

| number of bytes in num\_bytes. msg is the debugging string for

| the printout. Return the integer.

|

| GetReserved(self, num\_used)

| Construct a string of nul characters of such length to pad a

| command to one less than the packet size (leaves room for the

| checksum byte.

|

| Initialize(self, com\_port, baudrate, address=0)

|

| PrintCommandAndResponse(self, cmd, response, cmd\_name)

| Print the command and its response if debugging is on.

|

| Reserved(self, num\_used)

|

| ResponseStatus(self, response)

| Return a message string about what the response meant. The

| empty string means the response was OK.

|

| SendCommand(self, command)

| Sends the command to the serial stream and returns the 26 byte

| response.

|

| SendIntegerToLoad(self, byte, value, msg, num\_bytes=4)

| Send the indicated command along with value encoded as an integer

| of the specified size. Return the instrument's response status.

|

| StartCommand(self, byte)

|

| ----------------------------------------------------------------------

| Data and other attributes defined here:

|

| convert\_current = 10000.0

|

| convert\_power = 1000.0

|

| convert\_resistance = 1000.0

|

| convert\_voltage = 1000.0

|

| debug = 0

|

| highest\_register = 25

|

| length\_packet = 26

|

| lowest\_register = 1

|

| modes = {'cc': 0, 'cr': 3, 'cv': 1, 'cw': 2}

|

| to\_ms = 1000

FUNCTIONS

Register(pyclass=<class dcload.DCLoad>)

Unregister(classid='{943E2FA3-4ECE-448A-93AF-9ECAEB49CA1B}')

out = write(...)

write(str) -> None. Write string str to file.

Note that due to buffering, flush() or close() may be needed before

the file on disk reflects the data written.

DATA

division = \_Feature((2, 2, 0, 'alpha', 2), (3, 0, 0, 'alpha', 0), 8192...

nl = '\n'