B&K Precision 8500 DC Load Python Library

Table of Contents

Introduction

We provide a python library, `dcload.py`, that can provide programming access to a B&K DC load. The library is only supported for use on Windows computers.

There are two ways you can use this library:

1. You can use it to access a DC load from your own python programs. A `DCLoad` object is provided whose interface allows access to various features of the load. You may be able to use this method of access on non-Windows computers, but this is not supported by B&K.
2. You can use the `dcload.py` file to provide a COM server that provides access to the load. Using COM allows you to access the load using programming languages other than python, such as Visual Basic or Visual C++. COM is only available on Windows platforms.

NOTE: In this document, COM refers either to Component Object Model (a Microsoft programming technology) or the communications port on a PC (usually, COM1, COM2, etc.). The usage should be clear from the context.

Prerequisites

To use this library, you must have `python` and the `pyserial` library installed on your computer. If you wish to use the COM server, you must also have the `pywin32` library installed. Please see the appendices for how to get and install these tools.

Why a Library is Useful

The native programming interface to the DC loads is fairly low-level. It involves sending 26 byte commands and receiving 26 byte responses from the instrument. We'll demonstrate this interface in this section. First, it will demonstrate that your computer can talk to the instrument. Secondly, you'll likely see the need for a "higher-level" interface.

The following material assumes you're at a Windows command line.

Use your favorite editor (if you don't have one, you can use `notepad.exe`) to create a file called `test.txt` with the following single line:

```
import serial
```

Save the file, then type `python test.txt` at the command line. There should be no output and a command prompt should then be printed. If you see the message

```
ImportError: No module named serial
```

then the `pyserial` package wasn't installed correctly.

This demonstrates that you can create a python script and run it. There is no need for the file's name to have any special suffix.

Now we're ready to talk to the DC load. We'll assume you have the IT-E131 or IT-E132 interface installed between the computer and the DC load and have the driver installed if you are using the USB device. Make sure the DC load is powered on. Refer to the DC load instruction manual to set the baud rate of the DC load to the value you wish to use.

We will create a single command to the DC load that puts it into remote mode. This task illustrates many of the things that need to be done to talk to the instrument.

Create the following script and call it `serial.txt` (cut and paste is the fastest way to create it):

```
# Set DC load to remote mode.

import serial
length_packet = 26  # Number of bytes in a packet

def DumpCommand(bytes):
    assert(len(bytes) == length_packet)
    header = " "*3
    print header,
    for i in xrange(length_packet):
        if i % 10 == 0 and i != 0:
            print
            print header,
        if i % 5 == 0:
            print " ",
        s = "%02x" % ord(bytes[i])
        if s == "00":
            s = chr(250)*2
        print s,
    print

def CalculateChecksum(cmd):
    assert((len(cmd) == length_packet - 1) or (len(cmd) == length_packet))
    checksum = 0
    for i in xrange(length_packet - 1):
        checksum += ord(cmd[i])
    checksum %= 256
    return checksum

def main():
    port = 3  # COM4 for my computer
    baudrate = 38400
    sp = serial.Serial(port, baudrate) # Open a serial connection
    # Construct a set to remote command
    cmd  = chr(0xaa) + chr(0x00) + chr(0x20)  # First three bytes
    cmd += chr(0x01) + chr(0x00)*(length_packet - 1 - 4)
    cmd += chr(CalculateChecksum(cmd))
    assert(len(cmd) == length_packet)

    # Send command to DC load
    sp.write(cmd)
    print "Set to remote command:"
    DumpCommand(cmd)

    # Get response from DC load
    response = sp.read(length_packet)
    assert(len(response) == length_packet)
    print "Response:"
    DumpCommand(response)

main()
```

The last line in the script calls the function `main()`. The first three lines of the `main()` function set up a serial port to talk to. This serial object is created for us by the `pyserial` module we installed. Note that in `pyserial`, the first COM port on your computer is numbered 0; thus, if you're using COM1 on your PC, you'll set the `port` variable to 0.

The next five lines construct the string that we will send to the DC load. The `chr()` function creates a single character that has the ASCII value of the argument. The `+` symbols allows strings to be concatenated. The expression `chr(0)*a_number` creates a string of ASCII `0x00` characters whose length is `a_number`. The last character is the checksum of the previous 25 characters, calculated for us by the `CalculateChecksum()` function.

We use the `write` method of the `sp` object (the connection to the serial port) to send the command to the instrument. The `DumpCommand()` function prints the contents of the command to the screen in a hex format.

When a command has been sent to the instrument, you must always request the return data, which will always be another 26 bytes. This is also dumped to the screen.

Here are the results printed when this script is run:

```
Set to remote command:
      aa ·· 20 01 ··   ·· ·· ·· ·· ··
      ·· ·· ·· ·· ··   ·· ·· ·· ·· ··
      ·· ·· ·· ·· ··   cb
Response:
      aa ·· 12 80 ··   ·· ·· ·· ·· ··
      ·· ·· ·· ·· ··   ·· ·· ·· ·· ··
      ·· ·· ·· ·· ··   3c
```

The `·` characters represent the bytes with a value of `0x00`. This makes it easier to see the nonzero bytes in the string.

The first byte of a command is always `0xaa` and the second byte is the address of the DC load. The address should be set to 0. The third byte identifies the command "set to remote" and the fourth byte is a 1, which means enable remote mode. If the fourth byte was 0, this command would set the DC load to local mode.

The third byte of the response string is `0x12`, which means this is a packet that gives the status of the last command sent. The fourth byte is `0x80`, which means the command completed successfully.

On the DC load, you should see the `Rmt` annunciator turned on immediately after running the script. You will also see the `Link` annunciator light up while communications are going on, then blink out after a few seconds.

Press `Shift + Local` to return the DC load to local mode.

We've learned two key things about the DC load:

1. Commands are always sent as 26 byte packets.
2. For any command you send to the DC load, you must also request the return of a 26 byte packet. This returned packet will either be a status packet or contain the information which you requested -- for example, the power level currently set.

Get in the habit of looking at the LEDs on the IT-E131 or IT-E132 interfaces. Every command you send to the DC load should result in both the RX and TX LEDs blinking once. If this doesn't happen, something is wrong with the code, interface, or instrument.

If you peruse the DC load manual's programming section, you can see it will be tedious to construct all the commands as we did above. It would be a time saver to have a library do the low-level byte and bit manipulations for us. This was the rationale for developing the `dcload.py` module.

Using the Library from Python

The `pyserial` library abstracts the serial port. Thus, you may be able to use the `dcload.py` module for python programming on other platforms, such as Linux or Mac. However, B&K only supports use of the dcload.py module on Windows 2000 and later platforms.

At any time while using the library, you can execute

```
load.debug = 1
```

(where `load` is a DCLoad object defined in the module `dcload.py`) and you will turn on debugging printout. This causes the raw commands sent to and received from the DC load to be printed out. Set it back to 0 to turn debugging off.

Conventions

Many of the methods are getters and setters. This is object-terminology for methods that get or set state variables of a class (in this case the instrument, which is "hidden" inside the class). Usually, one value, such as a current, will be returned to you when you call the method. However, you should read the docstring of the method to understand what it returns. The docstring is a string that immediately follows the method name and is used to document the method.

All units used in the DCLoad methods are in SI; this means amperes, volts, watts, or ohms. You send values in these units and receive information back in those units. Internally, the DC load actually uses different units, but the library insulates you from that fact.

Return values

All of the library methods return strings. Methods that are used to set the state of the load will return the empty string if they are successful. Otherwise, the string contains an error message explaining what went wrong. Methods that return instrument settings will always return a nonempty string. It is up to you to parse or interpret that string correctly.

For robust code, you would want to check the return value of every such command. This can get tedious, so you may instead want to change the library to issue exceptions in these cases instead. This is beyond the scope of this note, so please refer to a python reference for how to do this.

Example

You can run the `client.py` script to see an example of use of the `dcload.py` module. From the command line, the command would be

```
python client.py obj p br
```

where `p` is the COM port number of the DC load and `br` is the baud rate the load is set to. This will produce output similar to (your numbers will differ)

```
Time from DC Load = Fri May 16 13:28:31 2008
Set to remote control
Set max current to 3 A
Set CC current to 0.2 A
Settings:
  Mode              = cc
  Max voltage       = 10.0
  Max current       = 3.0
  Max power         = 15.0
  CC current        = 0.2
  CV voltage        = 10.0
```

```
CW power             = 0.0
CR resistance        = 4000.0
Load on timer time   = 60000
Load on timer state = disabled
Trigger source       = immediate
Function             = fixed
Input values:
   6.392 V
   0.0 A
   0.0 W
   0x14
   0x0
Product info:
   8512

   1.78
```

TimeNow()

Returns a string with the current date and time. This is useful for data logging. Example:

```
Sun Dec 16 10:07:35 2007
```

The facilities in python's time module allow you to customize this string.

TurnLoadOn()

Turns the load on.

TurnLoadOff()

Turns the load off.

Remote or local control

SetRemoteControl()

Turns on the `Rmt` annunciator and puts the instrument into remote control. This means the only DC load keys that are functional are the up and down arrow keys and the `Shift + Local` key press.

SetLocalControl()

Returns control back to the front panel.

Maximums

These methods set the maximum values allowed to be set on the instrument. You can perform the same functions using the `:SYSTEM SET` menu from the DC load's front panel.

SetMaxCurrent(current)

Set maximum current in amperes. If you set the maximum current via the menu and front panel to a value larger than the instrument's capabilities, it will be set to the instrument's largest value. But if you try to do the same thing with the `SetMaxCurrent()` method, you'll get an error. Thus, you'll want to either use the `GetMaxCurrent()` method to verify that you set the correct value or examine the return value of the `SetMaxCurrent()` method.

GetMaxCurrent(sp)

Returns the currently-set maximum current in amperes.

SetMaxVoltage(voltage)

Analogous to `SetMaxCurrent()`.

GetMaxVoltage()

Returns the currently-set maximum voltage in volts.

SetMaxPower(power)

Analogous to `SetMaxCurrent()`.

GetMaxPower()

Returns the currently-set maximum power in watts.

Modes

SetMode(mode)

Sets the instrument to the specified mode, which is constant current, constant voltage, constant power, or constant resistance. mode must be one of the strings "cc", "cv", "cw", or "cr"; case is not important.

GetMode()

Returns one of the strings "cc", "cv", "cw", or "cr" indicating the currently-set mode.

Set mode parameters

SetCCCurrent(current_in_A)

GetCCCurrent()

Returns CC mode's current in A.

SetCVVoltage(voltage_in_V)

GetCVVoltage()

Returns CV mode's voltage in V.

SetCWPower(power_in_W)

GetCWPower()

Returns CW mode's power in W.

SetCRResistance(resistance_in_ohms)

GetCRResistance()

Returns CR mode's resistance in Ω.

Transient operations

SetTransient(mode, A, A_time_s, B, B_time_s, operation="continuous")

mode must be one of "cc", "cv", "cw", or "cr". operation must be "continuous", "pulse", or "toggled". Please see the instruction manual for the meanings of the other terms and how the triggering works.

GetTransient(mode)

Returns a string containing the mode, A, A_time_s, B, B_time_s, and operation. The variable names are the same as those used in the `SetTransient()` method.

Battery testing

SetBatteryTestVoltage(min_voltage_in_V)

GetBatteryTestVoltage()

Returns the minimum voltage in volts for the battery test.

Load On Timer

SetLoadOnTimer(time_in_s)

GetLoadOnTimer()

Returns the load on timer time in seconds.

SetLoadOnTimerState(enabled=0)

Set enabled to 1 to turn the load on timer state on. Set it to 0 to disable load on timer.

GetLoadOnTimerState()

Returns 0 (disabled) or 1 (enabled).

SetCommunicationAddress(address=0)

The address must be an integer between `0x00` and `0xFE`. Note: this address is currently unsupported and should always be set to 0.

Local control

EnableLocalControl()

Local control means the keys on the front panel work.

DisableLocalControl()

Local control disabled means only the up/down arrow keys work and the `Shift + Local` key (which returns the instrument to local control).

Remote Sense

SetRemoteSense(enabled=0)

Set enabled to 1 to turn on remote sensing. Set it to 0 to turn remote sensing off.

GetRemoteSense()

Returns 1 if remote sensing is enabled, 0 if not.

Trigger

SetTriggerSource(source="immediate")

The three choices for source are "immediate", "external", and "bus". "immediate" means a trigger occurs when the front panel `Shift + Trigger` is pressed. "external" means a trigger occurs when a TTL high signal longer than 5 ms reaches the trigger terminals on the back panel. "bus" means a software trigger is received (see the `TriggerLoad()` method).

GetTriggerSource()

Returns "immediate", "external", or "bus".

TriggerLoad()

Trigger the DC load by a software signal.

Save/recall Settings

SaveSettings(register=0)

Save the instrument's settings in the indicated register. register must be between 1 and 25 inclusive.

RecallSettings(register=0)

Recall the instrument's settings in the indicated register. register must be between 1 and 25 inclusive.

Functions

SetFunction(function="fixed")

function must be "fixed", "short", "transient", or "battery". Note "list" is not included, although it would not be much extra software work to add it.

GetFunction()

Returns "fixed", "short", "transient", or "battery".

GetInputValues()

Returns a string containing the voltage, current, power, op state, and demand state. The first three values are what are currently being displayed on the instrument's display panel. The last two are one byte and two byte integers (as hex strings), respectively, that contain coded information about the instrument's state. Please see the user manual for details.

GetProductInformation()

Returns a string containing the model number, serial number, and firmware version.

Using the COM Server

Using the COM server will only work on Windows computers. You'll need to install python, pyserial, and pywin32; please refer to the appendices for instructions.

Here's a summary of the things we'll do in this section:

1. Connect a DC load to the computer.
2. Register the COM server.
3. Run a python script to access the DC load via the COM server.
4. Use Visual Basic to access the DC load via the COM server

Connect the DC load

Connect a DC Load using either a serial port on your computer or the USB to serial device. If you use the USB to serial device (IT-E132), you'll need to install the PL-2302 driver on the CD that came with the IT-E132 device.

Make sure the DC load is powered on.

Register the COM server

Registering the COM server means telling Windows about the server. This only needs to be done once and it will be remembered across power cycles. One condition, however, is that the `dcload.py` file must not be moved or deleted; otherwise, the functionality will not be accessible through COM.

The easiest way to register the COM server is to open an Explorer window, navigate to the directory containing the python files in this software package, then double-click on the `dcload.py` file. A DOS window will open momentarily, then disappear, probably too fast for you to see what happened.

If instead you wish to see what happens during registration, open a DOS window and go to the directory that has the python file `dcload.py` that came with this software package.

Execute the command `python dcload.py`. You should see the message

```
Registered: BKServers.DCLoad85xx
```

This means the COM server has been registered with the operating system. Now an application can request a connection to a COM server named `BKServers.DCLoad85xx` and the operating system will start the `dcload.py` script when such a connection is requested.

If you wish to unregister the COM server, execute the command

```
python dcload.py --unregister.
```

You'd unregister, for example, if you wanted to move the `dcload.py` script somewhere else, then re-register it.

Use python to access the COM server

In a DOS window, go to the directory that has the python file `client.py` that came with this software package. Run the command `python client.py com p br`. Here, `p` is the COM (RS-232 communications) port the DC load is connected to and `br` is the baudrate. You should see output similar to the following (of course, your numbers will differ):

```
Time from DC Load = Fri May 16 13:28:31 2008
Set to remote control
Set max current to 3 A
Set CC current to 0.2 A
Settings:
  Mode                 = cc
  Max voltage          = 10.0
  Max current          = 3.0
  Max power            = 15.0
  CC current           = 0.2
  CV voltage           = 10.0
  CW power             = 0.0
  CR resistance        = 4000.0
  Load on timer time   = 60000
  Load on timer state  = disabled
  Trigger source       = immediate
  Function             = fixed
  Input values:
     6.392 V
     0.0 A
     0.0 W
     0x14
     0x0
  Product info:
     8512

     1.78
```

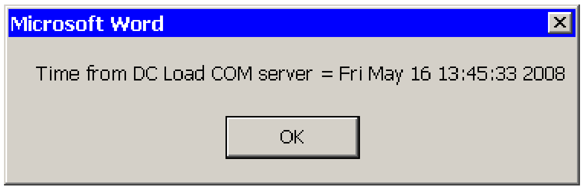
Use Visual Basic to access the COM server

If you have Microsoft Office installed on your computer, you can use the Visual Basic programming environment for the following example. This works with Word, Excel, PowerPoint, etc. If you have Visual Basic 6, the same code should run identically.

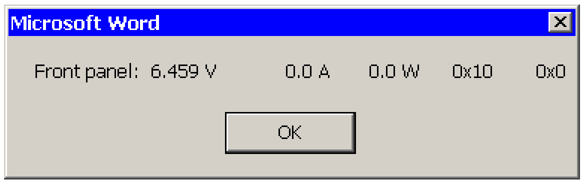
Open Word, then press `Alt-F11` to get into the Visual Basic editor. Enter the following function:

```
Sub dcload()
    ' Use DC Load's python COM server from Visual Basic
    Set server = CreateObject("BKServers.DCLoad85xx")
    port = 3      'COM3
    baudrate = 38400    '38400bps
    response = server.Initialize(port, baudrate)
    ' Get the time and display it in a message box
    response = server.TimeNow()
    msg = "Time from DC Load COM server = " + response
    MsgBox msg
    ' Get the DC load's front panel display
    response = server.GetInputValues()
    MsgBox "Front panel:  " + response
End Sub
```

Note you may need to change the `port` and `baudrate` variables. Put the editor's cursor in the `dcload()` function, then press the `F5` button to run the function. You'll see two message boxes similar to:



Time from DC Load COM server = Fri May 16 13:45:33 2008 and



```
Microsoft Word

Front panel:  6.459 V        0.0 A     0.0 W     0x10      0x0
```

The first message box demonstrates that the Visual Basic code is talking to the python COM server and the second message box demonstrates that the Visual Basic code talked to the DC load using the COM server.

*Note: If you only see the first message box and Word freezes, it is most likely because the DC Load is not set on “Remote” operation. Be sure to turn on “Remote” operation by sending the remote command “0x20” and “0x01” for bytes 3 and 4. You can quickly do this by calling the `SetRemoteControl()` method from the `dcload.py` library (i.e. *response* = *server.SetRemoteControl()*). For more references on command codes, see pages 54-74 in the user manual. Once set, restart Word and run script again.

Similar code should work from any Windows programming language that supports talking to COM servers.

Troubleshooting

All my commands are failing

If you get the message `Command cannot be carried out` (i.e., the `0xB0` status byte), it's possible that you haven't set the instrument to remote mode. Execute `SetRemoteControl()` before trying to execute your commands.

Make sure you're using the correct COM port number on your computer. Suspect this if the LEDs on the IT-E131 or IT-E132 interface boxes do not light up when you're sending commands.

Serial interface seems to be locked up

You've tried to send numerous different commands to the DC load, but none of them work -- your script hangs and needs to be stopped with ctrl-C.

First, check the LEDs on the IT-E131 or IT-E132 interface box. If you see the RX LED light, you know your command is reaching the instrument. In this case, make sure the DC load's internal address (under the `:CONFIG` menu) is set to 0.

If you see no blinking LED, suspect the interface box or the computer/driver.

DC load's front panel seems locked up

If you are using the short function or battery test function, there is no indication on the display that this is so. Press `Shift + Short` or `Shift + Battery` to see if the front panel starts responding again.

Python References

If you're an experienced programmer, you can learn python by reading the tutorial that comes with python (it's in the Doc directory of your python installation). You will likely be writing useful python programs in an hour or two.

There are numerous introductory python books available. For a beginning programmer, Lutz's *Learning Python* might be appropriate.

Other references are:

Martelli, *Python in a Nutshell*, published by O'Reilly.

Lutz, *Programming Python*, published by O'Reilly.

Beazley, *Python Essential Reference*, published by New Riders.

Appendices

Introduction:

For convenience, included in the “python.zip” file that can be downloaded from www.bkprecision.com are Python 2.5.2, pyserial 2.4, and pywin32 2.12. These are the 3 executable files required to use the python library and the COM server. If user decides to install using the provided files, please skip Step 1 in Appendix 1, Appendix 2, and Appendix 3. If user rather download the executable files themselves to obtain the latest versions, please follow all the steps as indicated in all three appendices in the following.

Appendix 1: Getting and Installing Python

Step 1:

Go to http://www.python.org and get the current production version of python. The installer is a Windows executable -- all you need to do is run it. You can accept the default choices if you wish.

Step 2:

Once python is installed, you need to add the python installation directory to your `Path` environment variable. On Windows XP, this is done by going to My Computer and right clicking to choose Properties. Click on the Advanced tab and then click the Environmental Variables button. The method for other versions of Windows can be different.

Step 3:

Open a DOS window (one way is to click on the Start button, select Run, then type in `cmd.exe`). Type `python` and press the Enter key. You should see something like the following:

```
Python 2.5.1 (r251:54863, Apr 18 2007, 08:51:08) [MSC v.1310 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

If you see this, python is installed correctly. This is known as the python interactive interpreter. Press `ctrl-z` and return to exit the python interpreter.

Appendix 2: Getting and Installing pyserial

Step 1:

Go to http://pyserial.sourceforge.net and download the `pyserial` package.

Step 2:

Unpack it in a convenient directory, then open a DOS window and `cd` to that directory. Type the command

```
python setup.py install
```

Step 3:

This will install pyserial. You can verify the installation by starting python, then typing `import serial` as follows:

```
Python 2.5.1 (r251:54863, Apr 18 2007, 08:51:08) [MSC v.1310 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import serial
>>>
```

If you see no error message after the `import serial` line, your `pyserial` installation was successful. Press `ctrl-z` and return to exit the python interpreter.

Appendix 3: Getting and Installing pywin32

Step 1:

Go to http://wiki.python.org/moin/Win32All and follow the links to the SourceForge repository.

Step 2:

Select the executable with the python version that matches the version of python you installed. Run the executable to install `pywin32`.

Step 3:

You can verify the installation by starting python, then typing `import win32com` as follows:

```
Python 2.5.1 (r251:54863, Apr 18 2007, 08:51:08) [MSC v.1310 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import win32com
>>>
```

If you see no error message after the `import win32com` line, your `pywin32` installation was successful. Press `ctrl-z` and return to exit the python interpreter.