CW High Voltage Control
S. Heppelmann Version 1

System configuration

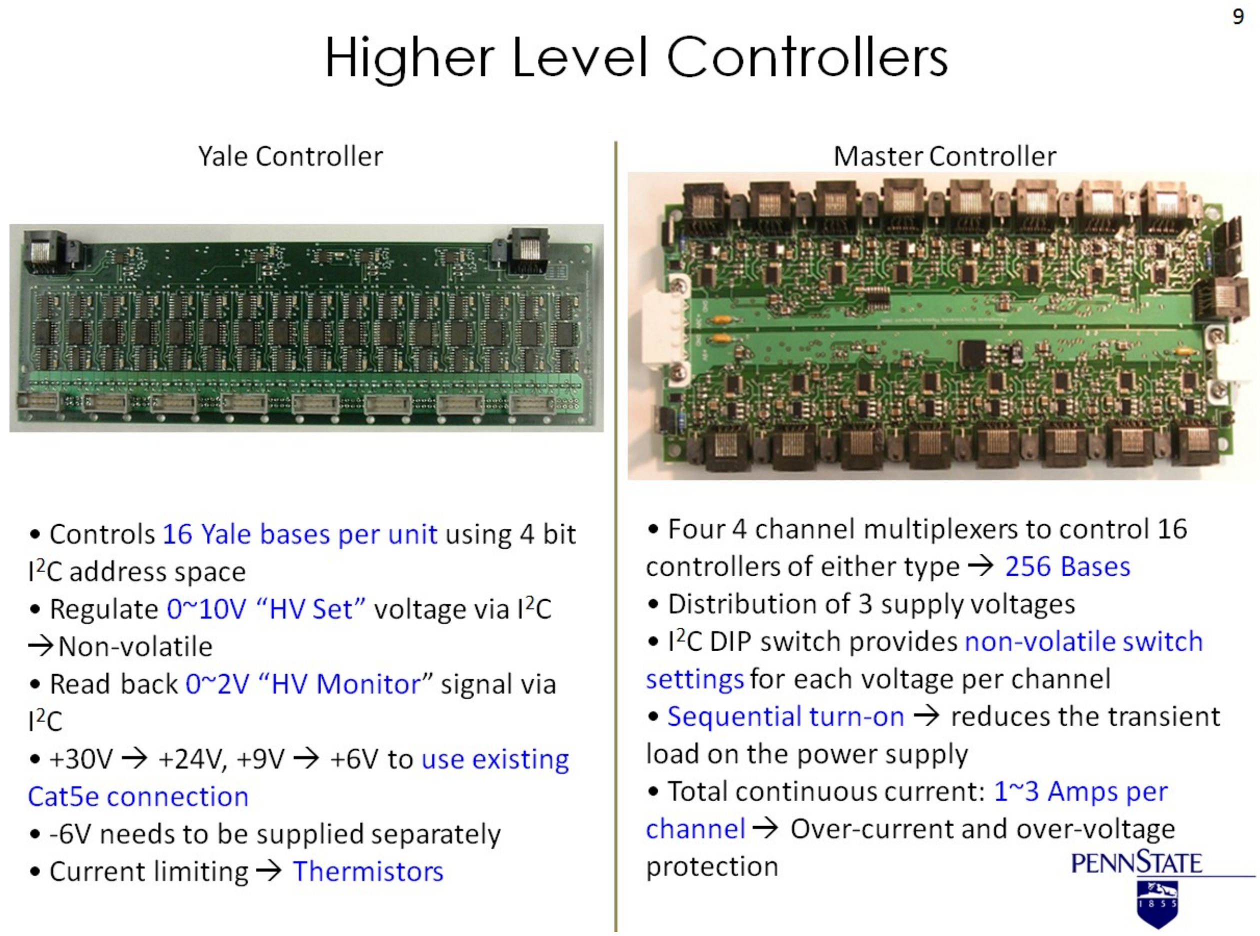


Figure 1: Len's FMS Review Talk

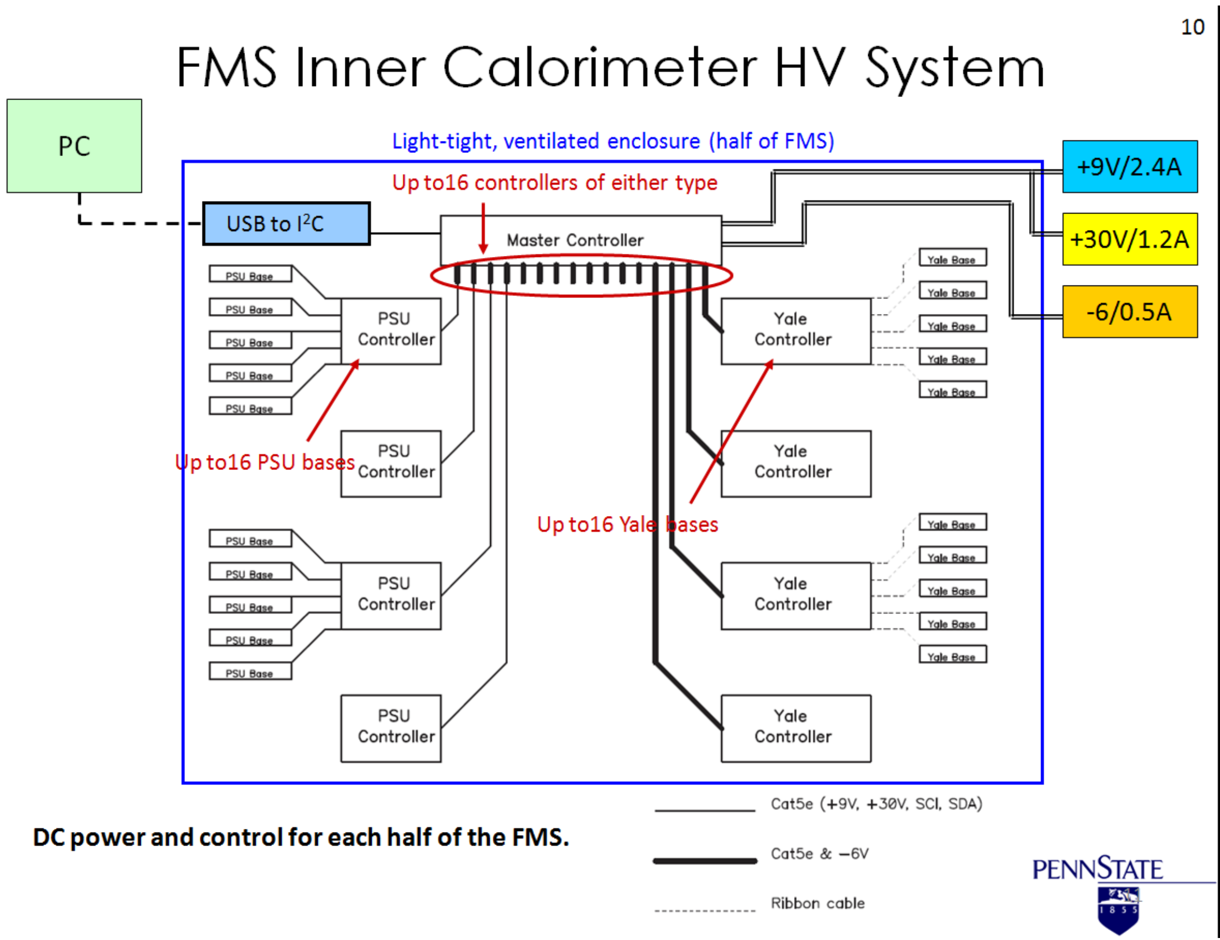


Figure 2: Len's FMS Review Talk.

Using CW high voltage control program "console.exe" is a windows program. The example below involves a windows XP environment with Cygwin running a bash shell.
There are three types of commands the the I2C console program

There are three types of input commands for the console program:

1) Raw I2C Read and Write

- Format: 4 fields of 1 hex byte

  [W/R] [ADDR] [CNT] [Repeat]
  (string of CNT data bytes for W/R=1)

  - [W/R] = 0 for read command and 1 for write command
  - [ADDR] = I2C Byte address
  - [CNT] = No of data words read or written
  - [Repeat]=0 Normal ; N for repeat of command N times (For debugging hardware)

2) Execute script files "@"

@Filename.txt [arg1] [arg2] [arg3]

- Run a script file from the debug directory.
- Arguments can be passed and will replace $0 , $1] , $2 etc in script file.
- Scripts can call scripts up to depth of 10.

3) Exclamation point commands "!"

![command name] [arguments]

for example:

assuming a path is established to a YALE or PSU board (including direct connection)

!rdac [ADDR]

- Read High Voltage settings for address ADDR.

!rdac [ADDR] [setvalue]

-Set Digital Pot and thus reset High Voltage

!PSUbase

-declare a PSU base

!YALEbase

- declare Yale base

!SETdevice [N]

-Talk to the N'th DevaSys device found on the USB bus. (dev=0 or dev=1)

!HVsave [ADDR]

-On Channel [ADDR] Copy the volatile voltage setting to non-volatile (powerup) memory.

!HVrestore [ADDR]

-On Channel [ADDR] restore the volatile HV to non-volatile value.

!HVGetTol [ADDR]

-Update the High Voltage Calculation Constants to account for rdac fabrication variation of ~10%.

!Sleep [MSEC]

- Sleep for MSEC milliseconds

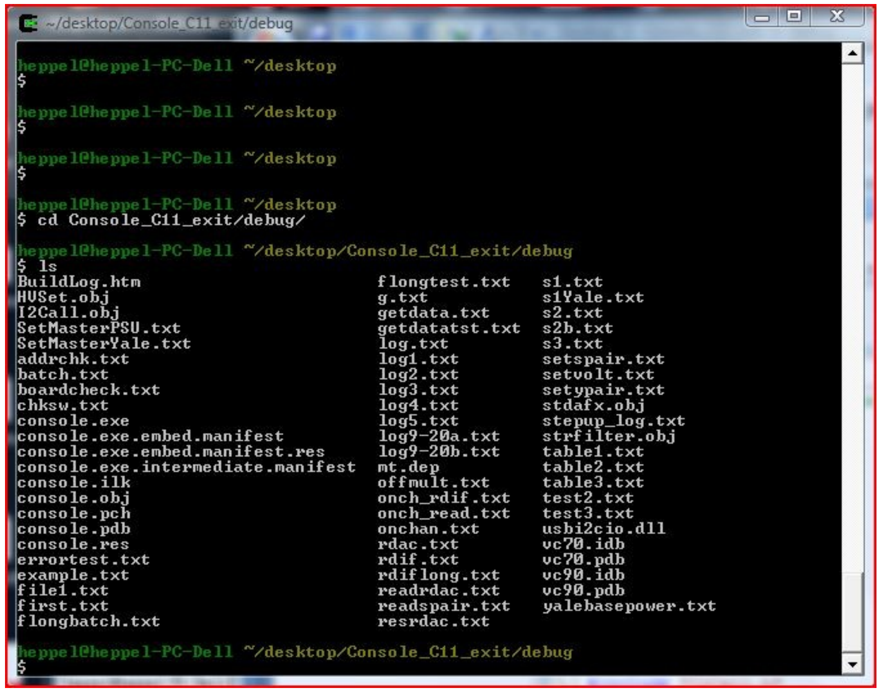
!ScriptRepeat [COUNT]

- sets a counter indicating the number of times that the next command script (ie. "@file.txt") will reopen and execute. A script with a Sleep command in it can be used to run periodically and log results. A counter exists at each level of "@file.txt". The counter at each level counts down toward zero each time the script file is reopened

-The nominal value for the counters at each level is zero. Counters only are increased when a "!ScriptRepeat [n]" is input.

Learn by example

1) Go to directory where executable lives.

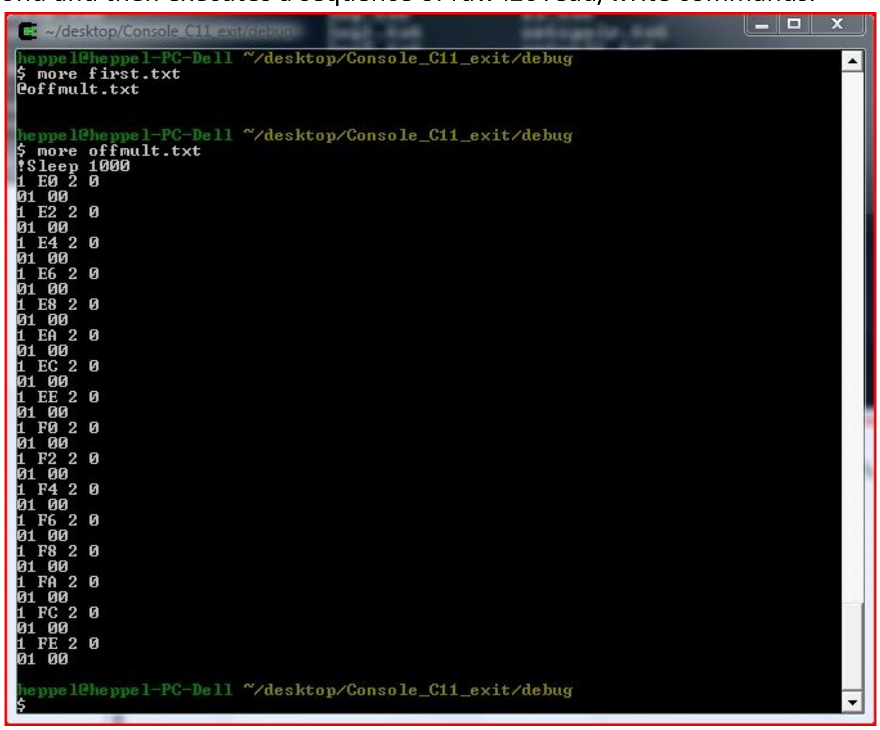


```
~/desktop/Console_C11_exit/debug
heppel@heppel-PC-Dell ~/desktop
$

heppel@heppel-PC-Dell ~/desktop
$

heppel@heppel-PC-Dell ~/desktop
$

heppel@heppel-PC-Dell ~/desktop
$ cd Console_C11_exit/debug/

heppel@heppel-PC-Dell ~/desktop/Console_C11_exit/debug
$ ls
BuildLog.htm                        flongtest.txt    s1.txt
HVSet.obj                           g.txt            s1Yale.txt
I2Call.obj                          getdata.txt      s2.txt
SetMasterPSU.txt                    getdatatst.txt   s2b.txt
SetMasterYale.txt                   log.txt          s3.txt
addrchk.txt                         log1.txt         setspair.txt
batch.txt                           log2.txt         setvolt.txt
boardcheck.txt                      log3.txt         setypair.txt
chksw.txt                           log4.txt         stdafx.obj
console.exe                         log5.txt         stepup_log.txt
console.exe.embed.manifest          log9-20a.txt     strfilter.obj
console.exe.embed.manifest.res      log9-20b.txt     table1.txt
console.exe.intermediate.manifest   mt.dep           table2.txt
console.ilk                         offmult.txt      table3.txt
console.obj                         onch_rdif.txt    test2.txt
console.pch                         onch_read.txt    test3.txt
console.pdb                         onchan.txt       usbi2cio.dll
console.res                         rdac.txt         vc70.idb
errortest.txt                       rdif.txt         vc70.pdb
example.txt                         rdiflong.txt     vc90.idb
file1.txt                           readrdac.txt     vc90.pdb
first.txt                           readspair.txt    yalebasepower.txt
flongbatch.txt                      resrdac.txt

heppel@heppel-PC-Dell ~/desktop/Console_C11_exit/debug
$
```

The *.txt files are script files that include legal commands for the program. When the program starts, the file "first.txt" is automatically executed.

The file **"first.txt"** includes one statement **"@offmult.txt",** that runs a script file of the same name. The script "offmult.txt" sleeps for 1 second and then executes a sequence of raw i2c read/write commands. These commands scan through the i2c address space, closing communication with each of 16 bases. The assumption is that the computer is directly connected to a YALE or PSU daughter board.

If the computer is plugged into a master controller then there is no path to a YALE or PSU daughter board until such a path is established. In that case, these commands do nothing.



```
~/desktop/Console_C11_exit/debug
heppel@heppel-PC-Dell ~/desktop/Console_C11_exit/debug
$ more first.txt
@offmult.txt

heppel@heppel-PC-Dell ~/desktop/Console_C11_exit/debug
$ more offmult.txt
!Sleep 1000
1 E0 2 0
01 00
1 E2 2 0
01 00
1 E4 2 0
01 00
1 E6 2 0
01 00
1 E8 2 0
01 00
1 EA 2 0
01 00
1 EC 2 0
01 00
1 EE 2 0
01 00
1 F0 2 0
01 00
1 F2 2 0
01 00
1 F4 2 0
01 00
1 F6 2 0
01 00
1 F8 2 0
01 00
1 FA 2 0
01 00
1 FC 2 0
01 00
1 FE 2 0
01 00

heppel@heppel-PC-Dell ~/desktop/Console_C11_exit/debug
$
```

So now we are ready to run the program "console.exe".

We see that the "first.txt" file ran and called "offmult.txt" as expected.

Output is also logged to a file "log.txt".

With the Enter prompt the program waits for one of three commands.

1) raw i2c command

2) @ command

3 ! command

Also you may enter "?" for help

```
~/desktop/Console_C11_exit/debug
$

heppel@heppel-PC-Dell ~/desktop/Console_C11_exit/debug
$

heppel@heppel-PC-Dell ~/desktop/Console_C11_exit/debug
$

heppel@heppel-PC-Dell ~/desktop/Console_C11_exit/debug
$

heppel@heppel-PC-Dell ~/desktop/Console_C11_exit/debug
$ ./console.exe
0 $0 1 $1
Number of ARgs=1
c:\Users\heppel\Desktop\Console_C11_exit\debug\console.exe
c:\Users\heppel\Desktop\Console_C11_exit\debug\

c:\Users\heppel\Desktop\Console_C11_exit\debug\first.txt
vers=772
Number of Devices=0
@offmult.txt
 offmult.txt[1,0]: Sleep 1000 milliseconds
offmult.txt[2,0]: Type=1,Addr=E0,N attempt=2, N r/w=-2;data: 1 0
offmult.txt[3,0]: Type=1,Addr=E2,N attempt=2, N r/w=-2;data: 1 0
offmult.txt[4,0]: Type=1,Addr=E4,N attempt=2, N r/w=-2;data: 1 0
offmult.txt[5,0]: Type=1,Addr=E6,N attempt=2, N r/w=-2;data: 1 0
offmult.txt[6,0]: Type=1,Addr=E8,N attempt=2, N r/w=-2;data: 1 0
offmult.txt[7,0]: Type=1,Addr=EA,N attempt=2, N r/w=-2;data: 1 0
offmult.txt[8,0]: Type=1,Addr=EC,N attempt=2, N r/w=-2;data: 1 0
offmult.txt[9,0]: Type=1,Addr=EE,N attempt=2, N r/w=-2;data: 1 0
offmult.txt[10,0]: Type=1,Addr=F0,N attempt=2, N r/w=-2;data: 1 0
offmult.txt[11,0]: Type=1,Addr=F2,N attempt=2, N r/w=-2;data: 1 0
offmult.txt[12,0]: Type=1,Addr=F4,N attempt=2, N r/w=-2;data: 1 0
offmult.txt[13,0]: Type=1,Addr=F6,N attempt=2, N r/w=-2;data: 1 0
offmult.txt[14,0]: Type=1,Addr=F8,N attempt=2, N r/w=-2;data: 1 0
offmult.txt[15,0]: Type=1,Addr=FA,N attempt=2, N r/w=-2;data: 1 0
offmult.txt[16,0]: Type=1,Addr=FC,N attempt=2, N r/w=-2;data: 1 0
offmult.txt[17,0]: Type=1,Addr=FE,N attempt=2, N r/w=-2;data: 1 0

Enter
 [W/R(1/0)] [Addr(hex)] [cnt:] [loop]
```

```
~/desktop/Console_C11_exit/debug

Enter
 [W/R(1/0)] [Addr(hex)] [cnt:] [loop]
?
raw i2c read/write          :  [0/1 for read/write] [i2c HEX addr] [N words] [nlo
op]
.... for write on N words   :  [data1] [data2] [data3]...[dataN]


Set or Read set base HV     :  !rdac [addr16 E0-FE] [optional value]
Read HV from addr           :  !radc [addr16 E0-FE]
Interpret as YALE base      :  !YALEbase
Interpret as PSU base       :  !PSUbase
Reset [addr16] to powerup   :  !HVrestore [addr16]

Each base is reached by setting dev[0-1]_chip[0-3]_chan[0-3]_base[0-15]

Set device to 0 or 1        :  !SETdevice [dev]
On   [Chip] 0 to 3          :  !
    Set [Chan] 0 to 3       :  !setctrl [chip] [chan]
    Set to [Chan]>3 to
    deselect chip (only one
    chip at a time)
read controller mem         :  !rmem [startaddr] [nbytes]
write controller mem        :  !wmem [startaddr] [nbytes] [string]
read controller id          :  !rctl
report crl chip registers   :  !chkchip [chip]
set or read power chip/chan:   !power [mask bit0:10V/bit1:30V/bit2:-6V]
run script called [name]    :  @name [param0] [param1]..
run next script [n] times   :  !ScriptRepeat [n]
sleep [m] msec              :  !Sleep [m]
check supply Voltage bits   :  !checkall 1
check for ctrl board ids    :  !checkall 2
i2c read set rdac array     :  !checkall 3
i2c read startup rdac array:   !checkall 4
read rdac array from file   :  !ReadTable [file]
dump rdac array to file     :  !DumpTable [file]
i2c write rdac set array    :  !Loadrdac
i2c read radc HV array      :  !checkall 5
i2c read rdac HV setpoints  :  !checkall 6
exit program                :  !exit

Enter
 [W/R(1/0)] [Addr(hex)] [cnt:] [loop]
```

Each base associated with a **YALE** or **PSU** card has one of 16 addresses. A commutation path determines which 16 base address space (which of the 32 YALE or PSU cards) is being accessed. The communication path is defined by setting

- dev (0-1)
- chip (0-3)
- chan (0-3)

The mapping is fully defined at this link location
http://drupal.star.bnl.gov/STAR/blog/heppel/2010/jun/25/deadcellsrun10179077#HighVoltage

There are two Master control cards used to control the FMS small cells, **dev=0** for the South Small Cells and **dev=1** for the North Small Cells.

To communicate through to a particular YALE or PSU daughter card (Figure 1 and Figure 2) a communication path must be established. The Master control card has 4 routing chips and each chip can enable 1 of 4 paths providing 16 distinct paths to one of the 16 attached YALE or PSU cards.

(Note: only one path should be established at a time)

This is how we would open up a path to dev=0, chip=3, chan=1

```
Enter
 [W/R(1/0)] [Addr(hex)] [cnt:] [loop]

!SETdevice 0

Enter
 [W/R(1/0)] [Addr(hex)] [cnt:] [loop]
!setctrl 3 2
```

After using this path, it is important to close it by setting the chip to a non-channel .
"!setctrl 3 4" to disable path through chip 3.

With the channel enabled, it is possible to talk to the i2c addresses on a **PSU** base or the similar control registers on the **YALE** mother board.
For each base there are multiple addresses registers defined.

Setting either of 2 values on the **rdac** chip (a variable resistor with 256 steps in resistance) can be done when the path has been mapped. The stored numbers are the volatile active voltages and the non-volatile reset values that survive repowering.

To set the volitile addr=0xE0 to value=0XAB

```
Enter
 [W/R(1/0)] [Addr(hex)] [cnt:] [loop]
!rdac E0 AB
```

To read back the volatile value from addr=0xE0

```
Enter
 [W/R(1/0)] [Addr(hex)] [cnt:] [loop]
!rdac E0
```

To save the volatile value to the non-volatile backup

```
Enter
 [W/R(1/0)] [Addr(hex)] [cnt:] [loop]
!HVsave E0
```

To set the volatile value to the non-volatile backup value

```
Enter
 [W/R(1/0)] [Addr(hex)] [cnt:] [loop]
!HVrestore E0
```

Low Voltage Control

The master boards are connected to 3 low voltage supplies.
V1=9V …… control bit 0 of hvmask ….. required for communications
V2=30V ……. control bit 1 of hvmask ….. required for high voltage
V3=-6V ……. control bit 2 of hvmask …… required for some additional features

The state of the low voltage for a particular YALE or PSU board is reflected by a 3 bit mask -> hvmask

To check the value of the mask for a particular board
!power

To reset the value of the mask for a particular board (and thus to turn on/off the low voltage power)
!power [hvmask]

To turn on all low voltage
!power 7

To turn off all low voltage
!power 0

Reading back voltage.

The voltage is read with a 8 bit ADC using the command which samples the actual high voltage.
!radc [addr]

To correctly interpret the voltage, we must declare the path as a YALE or PSU mother board before reading the ADC as above.

!PSUbase
or
!YALEbase
before
!radc [addr]

Local values of all rdac values.

The console program stores copies of all the **!rdac** setpoint values.
They are refreshed when we call
!checkall 3

This array of setpoint values can be save to a file
!DumpTable [file]

Or can be read back in from a file
!ReadTable [file]

These values can then be loaded into the CW system with
!Loadrdac