IUCF VME Data Acquisition System User Information

N. R. Yoder

Indiana University Cyclotron Facility

(Modified for KSU by Kevin Carnes)

1 Introduction

Since 1984, Indiana University Cyclotron Facility has used data-acquisition systems based on Unibus or Qbus VAX computers connected to Micro-programmable Branch Drivers (MBD) manufactured by BiRa Corporation. Newer, high-performance VAX workstations do not have the traditional Unibus or Qbus required by the MBD. Also, the MBD does not lend itself to easy programming by experimenters. There were several goals for replacement of the MBD; it should permit more flexible programming, be faster, allow greater distance between the CAMAC system and host analysis computer, and eliminate dependence on the host computer's bus hardware or general architecture and operating system. The VME-based data acquisition system described here provides a straightforward transition from the MBD systems, and continues to use the XSYS analysis software on the host computer, but is not limited to any particular host computer, operating system, or analysis software.

1.1 Hardware Design

The major components of the VME-based data-acquisition system are: (1) CAMAC subsystem; (2) VME frontend; (3) backend host computer. The basic VME frontend consists of a VME crate with one Motorola MVME167 68040 cpu module and one CES CBD-8210 CAMAC branch driver module. A second cpu module may be added for dedicated readout processing, and up to eight branch drivers may coexist on the VME bus, with each branch having up to seven CAMAC crates. VME data-acquisition modules may be used with no special interface; other subsystems such as FASTBUS or VICbus data modules may be used in addition to or instead of CAMAC if the appropriate driver modules are included in the VME crate. The MVME167 has serial, parallel, ethernet, and SCSI interfaces. One serial port is used for setting ethernet parameters and occasional debugging; all other communication with the host computer is by ethernet. Event data may be recorded on a 4- or 8-mm tape drive attached to the local SCSI interface, or on a tape or disk drive attached to the backend host computer. The backend host computer may be a VAXstation with XSYS software, or a UNIX system with software capable of processing event buffers from the VME frontend. An example of a VME data-acquisition system is illustrated in Figure 1.

A basic VME system consists of only one cpu and one CAMAC branch driver, and an ethernet connection. The backend host is responsible for taping.

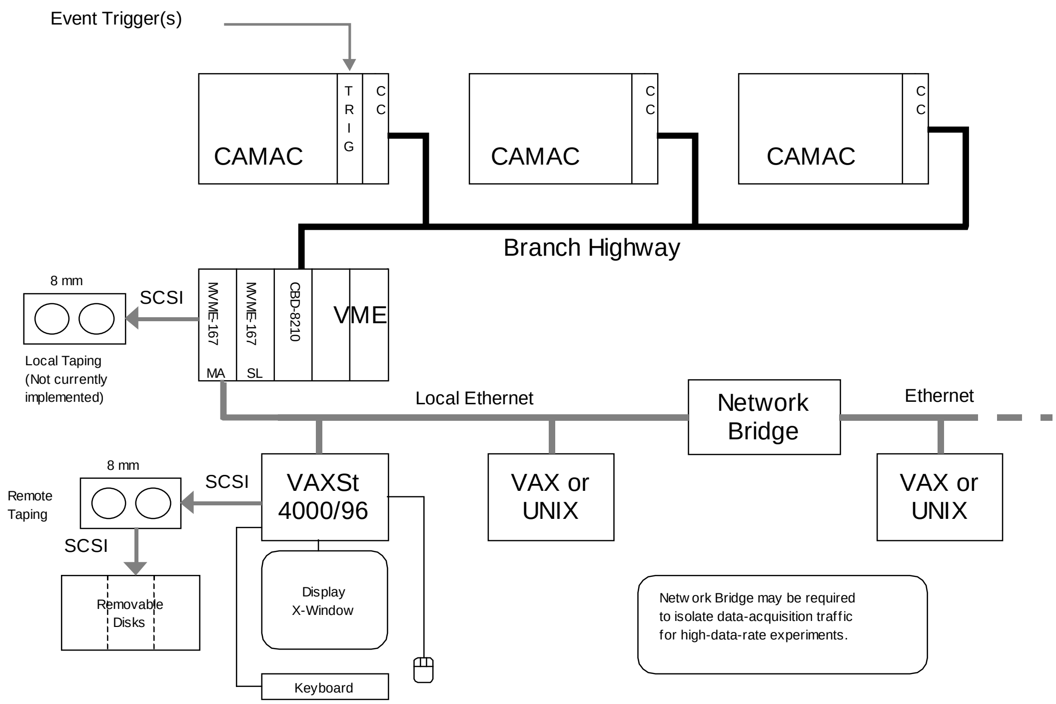

Figure 1. Typical VME data acquisition system.

1.2 Software Design

1.2.1 Real-time kernel

A real-time kernel, pSOS+, manufactured by Integrated Systems, Inc., is the foundation for the data-acquisition application program in the 68040 microprocessor(s). The components utilized for this application are:

- pSOS+m, real-time kernel with multiprocessing extensions
- pROBE+, debugger module
- pNA+, network module with TCP/IP stream socket services
- pREPC+, ANSI run-time library for C language applications

The kernel provides multi-tasking and inter-task communication services. A separate program in the host computer, XRAY+, works with pROBE+ for interactive debugging through the network. (Not available at KSU.) Any host computer with a TCP/IP socket library may be used to communicate through pNA+ to the application.

1.2.2 Application tasks

The data-acquisition application is organized for master and slave cpus, each with a set of tasks. The master cpu is responsible for run control, data buffer formatting and routing, and network communication. The slave cpu reads data modules (CAMAC) and builds event records.

<u>Master cpu:</u>

ROOT	Initializes system and starts all permanent master processes.
LISN	Listens to TCP/IP socket on predefined port number 6800 and creates one RC*nn*, RL*nn*, BS*nn*, or BR*nn* server task per client node request. Suffix indicates *nn*-th instance of a server task.
RC*nn*	Run-control and status.
RL*nn*	Run logging.
BS*nn*	Buffer send. Each sender can be configured to send a sample or subset of events.
BR*nn*	Buffer receive. Used to merge remote event streams, e.g. polarimeter events with primary data.
BFMT	Format buffers from the event queue and put buffer packets into a buffer queue. Byte swapping for VAX clients is done here.
BROU	Manage routing of formatted buffers to BS*nn* and BTAP tasks.
BTAP	Local 8mm taping (optional).

<u>Slave cpu:</u>

ROOT	Initializes slave and starts all permanent slave processes.
ECTL	Slave event and run control. Acts on messages from any RC*nn*.
KLOK	Timer signals for scaler reads and user-defined functions.
EVTS	Event-data readout builds event records and inserts data packets into an event queue.

In a single-cpu system, the master and slave ROOT processes are combined into one process. ROOT suspends itself after completing its function. Scheduling priorities of the data-handling processes are set so that EVTS is lowest, followed by BFMT and then BROU/BS*nn*/BTAP; LISN and RC*nn* have higher priority than data-handling processes.

Continuous beam and pulsed beam operations dictate different approaches for optimal throughput. The design permits the processes to fully overlap, but for pulsed beam it is better to let the event-data queue fill during beam on while suspending the formatting and sending processes until the beam is off. For example, a 1 MB event queue will accommodate a 1 second burst at the maximum branch highway rate of 900 kB/s, or a longer burst at a slower data rate. User-settable parameters control queue thresholds.

2 Run Control

Run control is implemented by one or more processes which provide interfaces between the frontend VME computer and a backend host for purposes of setting and displaying data-acquisition parameters and initiating actions which affect the run state and CAMAC devices.

2.1 Run Control Process

A VME run-control process *RCnn* is created by the *LISN* process whenever a request string beginning with "RC" is received after a TCP/IP connection is accepted. Several *RCnn* processes may be active at one time, up to a fixed limit, presently 6.

2.2 Connecting to Run Control Process

2.2.1 vrctl [-p *ipaddr*[:*port*]] [-o *reqstr*] [-c *command*]

vrctl executes on a VMS or UNIX host to provide bi-directional half-duplex communication between a terminal attached to the host and a VME run-control process. The default internet address *ipaddr* is obtained from the environment variable FRONTEND, if defined; the default *port* is 6800. The default request string *reqstr* is "RCTL EOT", which causes an <EOT> character (^D) to be appended to the final output line of each VME command. The <EOT> switches direction of communication and is not printed. If a *command* option is supplied, the command string is sent to the run-control process and input is not read from **stdin**; otherwise a prompt "rctl>" is supplied by vrctl; and input is read from **stdin**. The program is terminated when an exit or quit command is entered, or if the connection is broken, as by the shutdown command. A ^Z from a VMS host also terminates the program.

```
$ FRONTEND=="129.79.151.161"                    ! VMS
$ export FRONTEND=129.79.151.161 # UNIX
$ vrctl
Connected to 129.79.151.161:6800/1081
rctl>

$ vrctl -p 132.166.44.111 -c cnaf 3 7 0 0
```

2.2.2 telnet ipaddr [port]

Standard telnet may be used to connect to a run-control process; *port* must be 6800 and the request string "RC" must be entered within 5 seconds after a connection is established. The <EOT> option is not used because telnet provides a full duplex connection. The *RCnn* process issues the rctl> prompt.

```
$ telnet 129.79.153.164 6800
Trying...129.79.153.164
Connected to .
Escape character is '^]'.
RC
rctl>
```

2.2.3 User programmed function

A user-written function may issue run-control commands from a backend host by using a TCP/IP socket library to open a connection to the VME IP address at port number 6800. If the user function intends to operate in half-duplex mode, it should immediately send "RCTL EOT", and terminate each command line with a new-line character. Output lines will be separated by new-line characters and an <EOT> character following all output for each command. For full duplex mode, the function should send an initial "RC".

2.3 Run Status

The frontend is always in one of the following run states:

Disabled Unable to start data acquisition. This is the state immediately after software has been loaded or restarted, and after an error in starting data acquisition.

Off Enabled to start data acquisition. The program is in this state after an initialize or halt command.

On Data acquisition is active.

Held Data acquisition has been suspended, and the run may be resumed or halted.

Test Special test mode is in effect.

In addition to the run-state variable, there is a run-active flag variable which is true only in the **On** and **Test** states. Event data are read only when the run-active flag is on. Run status may be displayed by the show status command.

2.4 Commands

A run-control process accepts commands one line at a time. The command interpreter is sensitive to case, so enter commands as shown. Abbreviations are permitted for most keywords. Some commands may produce multiple output lines. The output for a single command is limited to the size of a fixed output buffer. For example, it is not possible to print the entire list of integer parameters with a single command.

Commands which affect the event-readout process are synchronized with the event-control *ECTL* and event-readout *EVTS* processes so that the command is always executed between processing events. Synchronized commands include *bcnaf, begin, cnaf, halt, init, init branch, init config, resume, show branch, suspend*. Some commands cannot be accepted when a run is active.

2.4.1 abort {run|test}

This command should not normally be used. It may be used to abort a **test**-mode run and return the run-state to **off**. When used with a normal run, it will clear the internal run active flag. Use of this command is not reccommended except in **test** modes.

2.4.2 bcnaf *b c n a f* [*d*]

Execute a CAMAC command on the specified branch *b*, crate *c*, slot *n*, subaddress *a*, function *f*, with data *d*. Numerical values may be specified in hexadecimal using a prefix of 0x, or octal using a prefix of 0. Response includes X, Q, and data for read commands. A nonexistent branch or off-line crate will cause an error message. Synchronized with event readout. Rapid execution of bcnaf concurrently with event-data readout is not recommended, and may lead to bus errors.

```
rctl>bcnaf 0 5 21 15 1
 X=1 Q=0 D=0x000002
```

2.4.3 begin [fill] [*n*]

Begin a run. If no run number *n* is supplied, the previous number is incremented by one. Begin will fail if the run state is not **off** and if there are no buffer-send *BSnn* processes. The option fill allows begin to fill the memory queues when there are no buffer-send processes or when there is already data in the queues. Begin performs a series of checks including verifying required on-line crates and trigger module configuration (default is BiRa 2206); the optional user-supplied begin code is executed. If successful, the run state is changed from **off** to **on**; otherwise it is set to **disabled**. See initialize.

```
rctl>begin 21
RC20- 21:03:20 (begin) #21
```

2.4.4 clear {all}

Clear VME run counters and scalers. This is automatically done by begin.

2.4.5 cnaf *c n a f* [*d*]

Execute a CAMAC command on branch 0. See bcnaf.

2.4.6 dump [l|w|b] *a* [*n*]

Dump VME memory in longword, word, or byte format, beginning at address *a* through *n* bytes. Longword is the default. The default radix is hexadecimal, and the default number of bytes depends on the format. An error message is printed if the address range is invalid. See also patch and probe.

```
rctl>dump w 180000
 00180000: 4e56 fff8 48e7 203c 202e 0008 222e 000c
 00180010: 207c 0019 2070 2448 207c 0019 21d4 2648
 00180020: 23c0 0019 203c 23c1 0019 2204 23ee 0010
 00180030: 0019 2034 42a7 4eb9 0018 929e 23c0 0019
 00180040: 2210 207c 0029 8000 23c8 0019 220c 2079
 00180050: 0019 203c 2268 0004 2868 0008 2028 000c
 00180060: 2068 001c 2a40 42ad 003c 23ec 0028 0019
 00180070: 2208 23d1 0019 2038 2068 0010 23e8 0004
```

2.4.7 exit

Exit from the run-control process and close the connection. This has no effect on the run state or other connections.

2.4.8 halt

Halt a run in progress. The run state is changed from **on** or **held** to **off**. Optional user code is executed and a scaler read is signalled. The default scaler read generates an event record with id number 50.

```
rctl>halt
RC20- 21:03:37 (halt) #21
```

2.4.9 help

A brief list of commands is printed. A question-mark (?) is an alias for help, and any command verb followed by a space and question mark prints additional syntax information.

```
rctl>help
command verbs:
    abort          bcnaf          begin          clear   cnaf
    dump           exit           halt           init    probe
    quit           restart        resume         set     show
    shutdown       suspend
For help on command syntax, type the verb followed by a space and ?
```

2.4.10 initialize [branch [*b*]|config|queue]

Initialize with no options initializes all branches and crates, verifies required on-line crates and executes optional user init code; if successful the run state is changed from **disabled** or **off** to **off**; if unsuccessful, the run state is set to **disabled**. Initialize branch issues a BZ to all branches or to branch *b*. Initialize config restores configuration parameters by re-executing user configuration code. Initialize queue clears data from the buffer queue if there are no buffer-send connections. The run state must be **disabled** or **off** before any initialize is accepted.

2.4.11 patch [l|w|b] *a v*

Patch the longword, word, or byte at address *a* with the value *v*. Longword is the default. Patch must be used with caution as incorrect use may corrupt or crash pSOS+ and the data-acquisition program. See dump and probe.

2.4.12 probe [l|w|b] *a*

Probe, i.e. read, the longword, word, or byte at address *a*. Longword is the default. Note that reading a non-memory device register may alter the state of that device. An error message is printed if the address is invalid. See dump and patch.

2.4.13 quit

Terminate the run-control process; equivalent to exit.

2.4.14 restart {warm|cold}

Restart pSOS+ and all processes. Restart cold triggers a download.

2.4.15 resume

Resume a run after suspend. The run state is changed from **held** to **on**. Required on-line crates and trigger configuration are verified as for begin, and optional user resume code is executed.

2.4.16 set BBclock [off]

Set the battery-backed clock from the pSOS+ operating clock. The battery-backed clock maintains time when pSOS+ is not running and when

VME power is off. The battery-backed clock may be switched off to save battery power before the cpu module is removed from service for an extended period.

2.4.17 set branch *b c ...*

Set the required on-line crate numbers *c...* for the specified branch *b*. This overrides configuration data in the user configuration code. If only the branch number is specified, then no crates are required to be online for that branch.

```
rctl>set branch 0 1 2
```

2.4.18 set buffer *size*

Set the buffer *size* in bytes for the buffer formatting process. This determines the maximum buffer size sent to the backend host. The default is 32256 bytes; buffers should be from 1024 to 65012 bytes.

2.4.19 set clock

Set the pSOS+ operating clock from the battery-backed clock.

2.4.20 set date yyyy.mm.dd [hh:mm:ss]

Set the operating clock date and time. Time-of-day is not changed if unspecified.

2.4.21 set date dd-mmm-yyyy [hh:mm:ss]

Set the operating clock date and time. This command uses VMS date format. Only the date is set if the time field is omitted.

2.4.22 set event {accum *n*[k]|size *n*}

Set the event queue accumulation threshhold to *n* bytes or *n* k-bytes; in fill mode, the event queue is allowed to fill until the threshhold is reached before the buffer formatting task is signalled. Set the maximum event size to *n* bytes; the maximum event size must not be exceeded by the user readout code. The size parameter is usually supplied by user configuration code or by system default.

2.4.23 set float *n v*

Set float parameter array element *n* to value *v*. User readout code has access to the float parameter array. The show configuration command prints the size of the float parameter array. All parameters are zero after program load or restart; user configuration code may preset values. Float parameters may be changed when a run is active.

```
rctl>set float 1 59.75
```

2.4.24 set int *n v*

Set integer parameter array element *n* to value *v*. User readout code has access to the integer parameter array. The show configuration command prints the size of the integer parameter array. All parameters are zero after program load or restart; user configuration code may preset values. Integer parameters may be changed when a run is active.

```
rctl>set int 3 450
rctl>set int 4 68
```

2.4.25 set run *n*

Set the run number to *n*. Begin will increment this number.

2.4.26 set mode {normal|acqisr|test {buffer|camac|user} [auto|debug]}

Set acquisition mode. Normal is the default; acqisr means that the event-data is to be read from within the interrupt service routine (ISR), which may be slightly faster in a one-cpu system. The test modes circumvent normal run control so that the data is actually read by the *BSnn* task. The mode options control the type of data; buffers are filled and sent with artificial data for test buffer; event records are filled by executing camac reads of the trigger-module id for test camac; the user-supplied readout is executed for test user. The auto option may be used with test user for auto (software) triggering; the debug option causes debugging output on the console serial port for test buffer.

2.4.27 set mode [-]buserr

Set or clear VME bus error handling. A bus error is commonly caused when an offline crate is accessed by user code. Certain other operations on the CBD-8210 occassionally cause bus errors. When bus error handling is disabled, a bus error causes the application to terminate and control is returned to the pROBE debugger. ***Bus error handling is always on for version 0.4 or later.***

2.4.28 set route [-]*nw gw*

Enter an internet gateway address *gw* for network address *nw* in the routing table. Addresses must be in the form *aaa.bbb.ccc.ddd*. A minus causes the gateway to be deleted from the routing table. This information is not usually required except in cases where a gateway is part of the route.

```
rctl>set route 0.0.0.0 132.166.51.254
```

2.4.29 set scaler timer *s*[.xx]

Set the interval timer for automatic scaler reads to *s.xx* seconds. Optional user scaler code performs the CAMAC reads. The minimum allowed scaler interval is 0.10 s. This command is equivalent to set timer 0 *s.xx*.

```
rctl>set scaler timer 10
```

2.4.30 set test [buffers|size] *n*

Set parameters for testing buffer send.

2.4.31 set time [yyyy.mm.dd] hh:mm:ss

Set the operating clock date and time. Date is not changed if unspecified.

```
rcl>set time 1993.09.01 19:30:00
```

2.4.32 set time [dd-mmm-yyyy] hh:mm:ss

Set the operating clock date and time. This command uses VMS time format. To set the VME clock from a VMS host clock:

```
$ vrctl -c "set time ''f$time()'"
```

2.4.33 set timer *n s*[.*xx*]

Set interval timer *n* for repeating at *s.xx* second intervals. Optional user code may be executed for each timer signal. The granularity is 0.01 second. Timer 0 is used for scaler read events; timer 1 generates a data event record; timers 2 and 3 execute user code without producing event records.

```
rctl>set timer 1 15
```

2.4.34 set title *text*

Set run title to the *text* string which follows the command. Quote marks are not necessary and if present will be retained; case is preserved. The title may be up to 80 characters in length. The default title is the system version string. The title is included in the run-control data buffers sent to the host.

```
rctl>set title E777 Detector test
```

2.4.35 set trigger {branch *n*|crate *n*|slot *n*|mask *n*|chan [-]*n* ...}

Set the trigger configuration. This information overrides parameters in user configuration code. The mask parameter is a longword with bits corresponding to enabled channels in a BiRa 2206 trigger module. Bits in the mask may also be set and cleared(-) using one or more chan parameters.

```
rctl>set trigger br 0 cr 5 sl 21 ch 4 ch 6 ch 10
rctl>set trigger ch -6 4
```

2.4.36 show BBclock

Show the time from the battery-backed clock.

```
rctl>show BBclock
BBclock = 1993.09.02 07:32:00  Thu
```

2.4.37 show branch [*b*]

Show branches and on-line crates by probing branch hardware. All attached branches are shown unless branch *b* is specified.

```
rctl>show branch
ONLINE BRANCH{CRATES}:  0{5}
```

2.4.38 show CBD *b* [*f*]

Show CES CBD-8210 CAMAC branch driver registers. Only one register is printed if *f* is specified. This information is for diagnostic purposes.

```
rctl>show CBD 0 4
 CBD[0,4] = 0x00ff
rctl>show CBD 0
```

```
CBD8210[0] csr=0xd83c btb=0x0020 glr=0x100002
   Am9519[1] sts=0xc7 isr=0x00 imr=0xff irr=0x00 acr=0x00
   Am9519[2] sts=0x87 isr=0x00 imr=0xff irr=0x00 acr=0x00
   Am9519[3] sts=0x87 isr=0x00 imr=0xff irr=0x00 acr=0x00
MVME167 intlev4=0x00001110  lbien=0x4004000e  lbistat=0x00000000
```

2.4.39 show configuration [all]

Show configuration parameters. User configuration code provides these data, which may be superceded by set commands.

```
rctl>show configuration
RC08-Required Branch{Crates}: 0{5}
        LAM Trigger: branch=0  crate=5  slot=21  mask=0x00030fe8
        buffer=32256 bytes  max_event=2048 bytes   num_scalers=256
        timers: scaler=0  slave1=0  slave2=0  slave3=0  (sec)
        user param: num_int=1024   num_flt=128
        test param: buffers=512   bytes/event=512
```

2.4.40 show connections [*]

Show TCP/IP connections and corresponding server process names. Disconnected connections are also printed if the command includes an asterisk.

```
    rctl>show connection
    RC08-07:07:46  (132.166.44.111)
    no. name  client_address:port  time_of_connect disconnect   status
 buf_cnt
     1. RC01  132.166.42.3:1047  Sep 01 17:49:40                 0000
 5
     9. RC08 129.79.151.24:1541  Sep 02 07:02:23                 0000
 6
```

2.4.41 show date

Show operating date and time.

2.4.42 show events [*n1* [*n2*]]

Show event counters by event-id number for the specified index range. All non-zero event counters are printed if no range is specified. The final event counter array is sent after a halt event.

2.4.43 show errors [CBD]

Show error log of CBD-8210 induced VME bus errors.

2.4.44 show fifos [*]

Show the status of buffer and event memory queues. A summary line for each packet in the buffer queue is also printed if the asterisk is present.

```
    rctl>show fifos
     queue    area:    end   insert   remove thrsh inuse  peak +in-out
 wait
     FEBUF   298000:  307fff 298000   298000  112k    0k    0k       0
     EVPKT   308000:  3fffff 308000   308000  248k    0k    0k       0
      febuf: put=0 get=0  evpkt: put=0 get=0
```

2.4.45 show float [*n1* [*n2*]]

Show float parameters for the specified index range. With no index range specified, all non-zero values are printed. The final float array is included in the buffer containing a halt event.

```
rctl>show float 1
RC08-Float parameters:
 [01]=       59.75
```

2.4.46 show int [*n1* [*n2*]]

Show integer parameters for the specified index range. With no index range specified, all non-zero values are printed. The final integer array is included in the buffer containing a halt event.

```
rctl>show int 3
RC08-Integer parameters:
 [003]=        450
rctl>sho int
RC08-Integer parameters:
 [003]=        450   [004]=         68   [1023]=          1
```

2.4.47 show scalers [*n1* [*n2*]]

Show scaler accumulations for the specified index range. All non-zero scalers are printed if no range is specified. The buffer formatting process accumulates scalar totals as it copies scaler event data to buffers. The final scaler accumulation array is included in the buffer containing a halt event.

2.4.48 show signals

Show the current data-event signal longword. Signal bits correspond to LAM and timer signals as well as internal synchronization signals. Bit assignments are defined by the union event_signal in daqdef.h

```
rctl>show signals
RC08-Event-signal mask=00000000
```

2.4.49 show status [all]

Show run status.

```
rctl>show status
Run Status: Disabled    Pending: None   Mode: Normal
        "IUCF-VME/DAQ MVME167(68040) v0.3 - Aug 10 1993 08:53:58"
        number = 1    starts=1    elapsed = 111.87
        started: Wed Sep 01 17:55:59 1993
        stopped: Wed Sep 01 17:57:51 1993
        buffers: formatted=20  routed=0  taped=0  words=287364
        events: control=4  scaler=0  data=1116  other=0
```

2.4.50 show time[s]

Show operating time. VME boot times are also shown for the plural command.

```
rctl>show time
RC08-Time = 07:20:40.63
rctl>show times
 Master time: Thu Sep 02 07:20:45 1993
 Master boot: Wed Sep 01 17:49:00 1993
 Slave  boot: Wed Sep 01 17:49:00 1993
 Clock ticks= 4870525  (100/s)
       utime= 2c859ecd   vmstime=00971eea 237ec480
```

2.4.51 show title

Show run title string. See set title.

```
rctl>show title
RC08-Title=IUCF-VME/DAQ MVME167(68040) v0.3 - Aug 10 1993 08:53:58
```

2.4.52 show version[s]

Show program version. Both the system version and the user version are shown, with compilation dates and times.

```
rctl>show versions
RC08-IUCF-VME/DAQ MVME167(68040) v0.3 - Aug 10 1993 08:53:58
user-<isis.vme data-acquisition program>
file-DAQLIB:USEREVTS.C Jul 16 1993 04:14:24
bcd date=19930810   vers=00000003
rctl>
```

2.4.53 signal [-][CBD *n*|LAM|IT2|IT4|ST0|ST1|ST2|ST3|NPA] ...

Set or clear one or more signals. Used for program testing and diagnosis. The CBD option affects the CBD-8210 interrupt status register.

2.4.54 shutdown

Shutdown pSOS+ and all processes. Use this command to return control to pROBE+ before reloading software. This is not synchronized with events and will interrupt a run in progress and break all network connections. Application code is not erased from memory and may be restarted using a remote debugger program.

2.4.55 suspend

Suspend a run. The run state is changed from **on** to **held**. Optional user suspend code is executed and a scaler read is signalled. See resume and halt.

3 Data-Acquisition Programming

3.1 Programming Tools

3.1.1 C Compiler

A cross compiler executes on a UNIX computer to generate 68040 machine code from ANSI C source files. The compiler output must be processed by the assembler.

3.1.2 Macro Assembler

A cross assembler executes on the UNIX computer to translate 68040 assembly source code to binary object files. Assembly source files use Motorola 680x0 syntax.

3.1.3 Linker

A linker executes on the UNIX computer to combine binary object files to produce a single loadable object file. Usually the load file is in Motorola hexadecimal format.

3.1.4 Make Utility

A make utility controls compiling, assembling, and linking as necessary. A standard make file is provided for building user code but may be modified if for instance, the user has additional code modules to be compiled and linked. See the system administrator for more details.

3.2 EVTS Process

3.2.1 userevts.c file.

The userevts.c file contains the following include files. Files beginning with "user" contain user supplied code to perform a specified function. Normally the userevts.c file itself is not modified, but only the user include files. However, if userevts.c or any of its include files are found in the user code directory, they will be processed by the make utility in place of the standard files. The following include files should <u>never</u> be modified:

```
#include "daqlib.h"
#include "daqdef.h"
#include "usrlib.h"
#include "usrdef.h"
#include "camac.h"
#include "usrdb.h"
```

3.2.1.1 user-supplied include files

The following include files contain user-specified code for run control and data acquisition. If any of these files is not found in the user code directory, a standard dummy file will be provided by the make procedure.

```
#   include "userconfig.h"
#   include "userinit.h"
#   include "userbegin.h"
```

```
#   include "userend.h"
#   include "userresume.h"
#   include "usersuspend.h"
#   include "userfunct.h"
#   include "userevents.h"
#   include "userscalers.h"
#   include "userstimer1.h"
#   include "userstimer2.h"
#   include "userstimer3.h"
#   include "useritrig2.h"
```

3.2.1.1.1 userconfig.h

Contains information on which CAMAC crate slots will generate LAMS, how often to read scalers, etc. Always required.

3.2.1.1.2 userinit.h

Executed as part of initialize. CAMAC branches and crates are initialized before this user code executes. Configuration parameters should not be changed in initialize code.

3.2.1.1.3 userbegin.h

Executed as part of a begin run. The trigger module is initialized before user code executes, if a BiRa 2206 is used.

3.2.1.1.4 userend.h

Executed as part of a halt run.

3.2.1.1.5 userresume.h

Executed as part of a resume run

3.2.1.1.6 usersuspend.h

Executed as part of a suspend run.

3.2.1.1.7 userevents.h

Executed when an event LAM occurs. If a BiRa 2206 trigger module is used, the code in userevts.c reads the trigger data and stores a number in *trigchan*. The user code uses a switch statement to select the event type.

3.2.1.1.8 userscalers.h

Executed when a scaler timer signal occurs and when halt or suspend is done. An event record is generated with longword byte-swapping and scaler-buffer attributes.

3.2.1.1.9 userstimer1.h

Executed when slave software timer 1 signal occurs. Timer signals are repetitive. An event record is generated.

3.2.1.1.10 userstimer2.h

Executed when slave software timer 2 signal occurs. Timer signals are repetitive. No event record is generated.

3.2.1.1.11 userstimer3.h

Executed when slave software timer 3 signal occurs. Timer signals are repitative. No event record is generated.

3.2.1.1.12 usertrig2.h

Executed when external trigger IT2 on the CBD-8210 receives a pulse. An event record is generated.

3.2.1.2 Include code examples

3.2.1.2.1 userconfig.h example

```
/**********************************************************************/
/* This file is included in the standard user configuration function.  */
/* It is executed once after the frontend processor is started, and    */
/* whenever an "init config" command is issued.                        */
/* This code is executed before any crates are initialized.            */
/* The purpose is to specify which crates are required and the         */
/* location of the trigger module.                                     */
/* No camac operations should be performed here.                       */
/* Configuration may be aborted by executing "return -1;"              */
/* Initialization will be done after successful configuration.         */
/**********************************************************************/
/* Static user parameters                                              */
/* User configuration code (userconfig.h) should set these as follows: */
/*  udb.usrverstr_p = "version";* user program version string          */
/*  udb.usrtitstr_p = "title";  * user initial run title string        */
/*  udb.maxevtsiz = size;       * if event size is > default(bytes)    */
/*  udb.btbmasks[b] |= (1<<c);  * for each required online crate       */
/*  udb.lamcrmask |= (1<<c);    * c is LAM trigger crate (always 1)    */
/*  udb.lamslmask |= (1<<m);    * m is LAM trigger slot# -1
/*  udb.trigbranch = b;         * specify branch for lam trigger       */
/*  udb.trigcrate = c;          * specify crate for lam trigger        */
/*  udb.trigslot = n;           * specify slot for trigger module      */
/*  udb.triglamn = n;           * specify slot for trigger lam         */
/*  udb.trigmask |= (1<<m);     * for each BiRa trig channel           */
/*  udb.st0tik = n;             * ticks for scaler timer (10msec units)*/
/*  udb.st1tik = n;             * ticks for slave timer 1              */
/*  udb.st2tik = n;             * ticks for slave timer 2              */
/* For BiRa2206, set trigbranch, trigcrate, trigslot, trigmask.        */
/* If LAM is jumpered to different slot, set triglamn = LAM slot.      */
/* For other trigger module, set trigmask=0; such trigger module must  */
/* return Q=? for A(0)F(8), and clear lam for A(0)F(10).               */
/* For no trigger module, set trigslot=trigmask=0                      */
/* Only one BiRa2206 is supported by built-in code; for multiple LAM   */
/* sources, follow the directions for general LAMs below.              */
/* A BiRa2206 and other LAMs may be simultaneously configured.         */
/*                                                                     */
/* For general LAM triggers from modules in multiple slots and crates, */
/* set lamcrmask and lamslmask to indicate which crates and slots may  */
/* produce LAM interrupts.  This is the standard KSU configuration.    */
/* If lamcrmask=0 and a BiRa2206 is not configured, then the default   */
/* crate mask is btbmasks and branch demand will be enabled for        */
/* those crates specified by btbmasks[trigbranch].                     */
/* All LAM triggers must be confined to a single branch.               */
/**********************************************************************/

/* Set parameters for example user configuration */

udb.usrverstr_p = "<Version 1>";
udb.usrtitstr_p = "<2D PSD Phillips ADC>";
udb.maxevtsiz = 0;            /* uses default if 0 */
udb.btbmasks[0] = (1<<1);
udb.lamcrmask = (1<<1);
udb.lamslmask = (1<<19) | (1<<20); /*allows for two different LAMs,   */
                                   /*here in slots 20 & 21 (slot#-1)  */
udb.trigbranch = 0;  /* udb.trig* parameters only for BiRa2206 useage */
```

```
udb.trigcrate = 0;
udb.trigslot = 0;
udb.triglamn = 0;
udb.trigmask = 0;
udb.st0tik = 500;              /* in 10msec ticks, e.g. 500 = 5 sec*/
                               /* This is the time interval for    */
                               /* automatic scaler reads           */
udb.st1tik = 0;
udb.st2tik = 0;
udb.st3tik = 0;
/* */
```

3.2.1.2.2 userevents.h example

```
/************************************************************************/
/* Get event data record.                                               */
/* Reads data from CAMAC and fills event record with id and data.      */
/* event id is preset to trigchan if using BiRa2206.                    */
/* BiRa2206 is handled outside of this code.                            */
/* Without BiRa2206, event id must be set manually.                     */
/************************************************************************/


int eventid;
if(tlamregs & (1<<19))          /* Check to see if LAM is from slot 20 */
   {
   eventid = 3;                 /* Set event number for EVENT in *.evl */
READW(0, 1, 20, 4, 0);          /* Read 7164 */
READW(0, 1, 20, 5, 0);          /* Read 7164 */
READW(0, 1, 20, 6, 0);          /* Read 7164 */
READW(0, 1, 20, 7, 0);          /* Read 7164 */
CNAFX(1, 20, 3, 11);            /* Clear data and hit register on 7164 */
   }
else if (tlamregs & (1<<20))    /* Check to see if LAM is from slot 21 */
   {
   CNAFX (1,21, 0, 10);         /* Clear Down Counter                  */
   eventid = 4;                 /* Set event number for EVENT in *.evl */
   HALTRUN;                     /* Halt the run.                       */
   }
ev_p->hdr.evtid += eventid;     /* Actually set EVENT number.          */
```

3.2.1.2.3 userbegin.h example

```
/************************************************************************/
/* User initialization of modules at beginning of run.                  */
/* This is called before data flow is enabled.                          */
/* A BiRa2206 is automatically initialized before this code.            */
/* To abort begin run, execute "return -1;" here.                       */
/************************************************************************/


CNAFX (1, 10, 0, 9);                 /* clear counter channel */
CNAFX (1, 10, 1, 9);                 /* clear counter channel */
CNAFX (1, 10, 2, 9);                 /* clear counter channel */
CNAFX (1, 10, 3, 9);                 /* clear counter channel */

CNAFX (1, 20, 3, 11);                /* clear Phillips 7164 ADC */
CNAFX (1, 20, 0, 26);                /* set enable LAM for 7164 ADC*/
CNAFX (1, 21, 0, 26);           /* set enable LAM for Bi-Ra 2204 DPR Counter*/
CNAFX (1, 30, 9, 24);                /* disable dataway inhibit */
```

3.2.1.3 Programming macros

In the following commands: b=branch address (always 0 at KSU)
c=crate address (always 1 at KSU)
n=slot in crate
a=address in a given module
f=CAMAC function code

QPRESENT(b)	True if CAMAC cycle Q=1
XPRESENT(b)	True if CAMAC cycle X=1
BCNAF(b,c,n,a,f)	Execute 16-bit CAMAC command
BCNAFL(b,c,n,a,f)	Execute 24-bit CAMAC command
BCNAF_Q(b,c,n,a,f)	Executes 16-bit CAMAC command and is true if Q=1
BCNAF_X(b,c,n,a,f)	Executes 16-bit CAMAC command and is true if X=1
BCNAFX(b,c,n,a,f)	Executes 16-bit CAMAC control command and slight delay
CNAF(c,n,a,f)	Executes 16-bit CAMAC command on branch 0
CNAFL(c,n,a,f)	Executes 24-bit CAMAC command on branch 0
CNAF_Q(c,n,a,f)	Executes 16-bit CAMAC command on branch 0 and is true if Q=1
CNAF_X(c,n,a,f)	Executes 16-bit CAMAC command on branch 0 and is true if X=1
CNAFX(c,n,a,f)	Executes 16-bit CAMAC control command on branch 0 and slight delay
QTMRDCH(c,n)	Read BR2206 for which channel triggered
QTMRCCH(c,n)	Read BR2206 for which channel triggered and clear channel
QTMCLCH(c,n,ch)	Clear BR2206 channel #ch
QTMCLAM(c,n)	Clear BR2206 LAM
QTMCLER(c,n,ch)	Clear BR2206 channel #ch and LAM
RESETEVENT	Reset buffer pointer back to beginning of buffer
FLUSHBUFFER	Flush buffer (send buffer to host)
HALTRUN	Halt run
READW(b,c,n,a,f)	Do CAMAC 16-bit read and store in buffer
READL(b,c,n,a,f)	Do CAMAC 24-bit read and store in buffer
READBW(b,c,n,a,f,max)	Do CAMAC 16-bit read and store in buffer until Q=0 up to max reads
READBL(b,c,n,a,f,max)	Do CAMAC 24-bit read and store in buffer until Q=0 up to max reads
WRITEW(b,c,n,a,f,d)	Do CAMAC 16-bit write of data word “d”
WRITEL(b,c,n,a,f,d)	Do CAMAC 24-bit write of data word “d”