Online framework for monitor and diagnostic routines

December 1, 1998

1 Framework Task Overview

A data monitor routine must provide some or all of the following services:

- Provide event data monitoring and analysis.
- Collect statistical measures over some time period, typically one run.
- Report errors to an error logger or Expert System.
- Be synchronized with the run sequence.

An online framework is a modification of the data monitor skeleton and has been defined to perform most of the common functionality of the many monitoring applications that will be developed for the Hera-B DAQ. The reasons to define a skeleton for general use in all control applications are:

- Minimize the amount of online software development required of the detector groups.
- Insure common interface specifications are followed.
- Insure that developments can be used by all detectors.
- Facilitate maintenance of the the code.
- Provide a common upgrade path.

The online frame has hooks onto which may be hung the routines needed to perform the specific functionality of the application. Dummy versions of these application specific routines are defined in a template which can be used by the application developer as the basis of his program.

The frame provides the following functionality:

- Interfaces the monitor routines to the Hera-B event manager.
- A message based interface to other processes.
- Run state management.

- Run time parameter presentation and setting.
- User's idle loop application.

A package of routines (event Tool Box ETB and Event Swap Package ESP) are supposed to be used in users routines to facilitate access to specific event record sub-blocks.

1.1 Dictionary of Terms

- **Consumer -** A process that reads event data from a shared memory partition.
- **emg -** The Hera-B event data manager.
- **ETB -** The Hera-B event Tool Box. This package facilitates access to data stored in the DAQ event format.
- **ESP -** The Hera-B event Swap Package. This package allows to swap bytes in the various framing of the event structure.
- **Partition -** A shared memory area containing event buffers (segments) managed by the emg package.
- **rpm -** The Hera-B online message passing facility[1].
- **smc -** The Hera-B State Machine Control package for the run control system[3].
- **par -** The Hera-B online process parameter package that makes a set of parameters of an application program available for modification or interrogation by other online processes.

1.2 Environment

The online frame runs under the standard c run-time environment of most Unix-like operating systems (Irix, Lynx and Linux are currently supported). In addition to the base system the ipc (interprocess communications) and BSD sockets (networking) packages are used extensively.

The online frame also requires the following Hera-B specific online packages:

- **rpm** the Hera-B messaging package.
- **emg** the Hera-B event manager.

- **par** the Hera-B process parameter package.
- **smc** the Hera-B state machine control package.

Note that although the **rpm** libraries must be available when data monitors are linked, the frame can run without **rpm** if the **-C** parameter has been specified and rpm cannot be initialized (*e.g.* because the network is down).

1.3 State Control

The online frame is usually controlled by the run control system. It can also run without its control if **-C** specified on the command line.

1.4 User Interface

The user interface to the frame is not yet well defined. This can be decided on soon to allow make uniform access available to all frame users. The state machine and the interprocess communication packages described above can and should be used by the user interface programs for controlling and monitoring.

1.4.1 Error reporting

Error reporting is provided by the *grr_error*() function. Errors reported by *grr_error*() are sent to the domain error logger and in the case of severe errors, to the global error logger as well. The detector mask which is sent with the error message to insure that the it is routed to the right places can be set at initialization time with the *grr_mask*() function. These routines are described in Section 5.

1.5 Left to do

The integration of the remote histogram package (**rhp**) hasn't been implemented yet. This will be done as soon as the package arrives or the users will use **rhp** interface in the routines described in Section 4.

2 Monitor Program Development

2.1 Writing the program

A dummy template of all the routines needed by the frame is located in /hb/daq/Online_Library/examples/dmon-user.c. This can be copied and modified to the specific needs of any control application.

2.2 Compiling & Linking

A template make file is included in /hb/daq/Online_Library/examples/dmon-make. In case of Solaris operating system one should define the variable $SOCKLIB as "-lsocket -lnsl". This make file compiles and links a program called 'dmon' with the frame and all the necessary online libraries. This can be copied and modified to build detector specific control applications.

2.3 Running in the Online Environment

The *dmon* routine is run with the following command line:

```
dmon [-H] [-C] [-S<size>] [-L<size>] [-p<file>] [-F<file>]
       [-A] [-G<partition>] [-D<comp>]
       [-T<trig>]  [<rpm-options>] [<SMC-options>] [-h]
```

The **-H** invokes the program help.

The **-C** flag disables control of the monitor process by the state machine. If **-C** is specified, data will be processed continuously, regardless of starting or stoppage of runs.

The *dmonuser_init*() and *dmonuser_halt*() routines are called at the frame start-up and termination correspondently.

The *dmonuser_begin*() routine is called at the beginning of every run (setting to the state **STANDBYE** or **RUN**) , and the *dmonuser_end*() routine is called when the frame goes neither to **STANDBYE** nor to **RUN** state.

The *dmonuser_loop*() is always called at the start-up of the program after *dmonuser_init*() completion. All next recalls of the *dmonuser_loop*() routine depend on the status returned by the routine.

The possible statuses are:

- **onevent** - the routine will be called on getting event.
- **always** - the routine will be callee all the time.
- **never** - the routine will never be called.

The default status is set to **onevent** on the frame start-up. That means that *dmonuser_loop*() will be called every time after an event has arrived.

If **always** is returned from the *dmonuser_loop*() then the routine will be called all the time if there are no RPM messages. In this mode the user has to take care of event reading. In this mode neither *dmonuser_begin*() nor *dmonuser_end*() is to be called if the state machine is being used.

The value **never** got from the *dmonuser_loop*() stops all the following calls of the routine.

If the online frame is unable to initialize **rpm** (usually because the name server is not running or not accessible) and **-C** is specified, the monitor will continue to run without **rpm**.

-S can be used for the event buffer size setting if events are to be read from the file (the default buffer size is 128Kb). In case of reading events from the shared memory partition the event buffer size is set to the value of its segment length.

-L can be used for the patrition segment size setting if the default size (1024 bytes) is larger then one that the target partition has.

If **-A** is used then all the events meeting **-D<comp>** and **-T<trig>** requirements wiil be reserved for the application.

-p<file> can be used for initial settings of the published parameters (see Section 5). If no file is specified, *dmon.par* is used.

-F<file> allows to read events from the files. If no file is specified, *dmon.data* is used. In this mode the state machine control is disabled.

The shared memory partition to be used is specified by the **-G<partition>** argument. If no partition is specified, "Hera-B DAQ" is used. The partition name containing your data may depend on you run configuration. The *dmonuser_event*() routine is called after getting event from the partition. Depending on the returned status the number of events or errors is to be incremented for the statistics.

The **-D<comp>** and **-T<trig>** options allow the user to request processing of certain components and/or triggers, respectively. **<comp>** is a decimal number corresponding to the component number (see component.h) and **<trig>** is a mask (specified in hex) with the usual meaning .

<rpm-options> refers to any of the **rpm** message package options that may be appropriate. They are described in the message system documentation [1]. These include **-Domain:<domain>** and **-Process:<process>**, the rpm domain and program name. The domain name specifies which name server will register the process name and thereby associates the process with a run. If no process name is specified explicitly, the command name is used. For **rpm** message routing the calls *rpm_bind*() and *rpm_dispatch*() are used in the frame. All message classes besides **PAR_CLASS**, **PUB_CLASS**, **SMC_CLASS** should be *bind* by the user in *dmonuser_init*() routine.

<SMC-options> refers to any of the **smc** message package options that may be appropriate. They are described in the **smc** documentation [3].If no options are specified the defaults will be used. The state transition table (STT) (contained in two files) should be in the working directory if **smc** is to be used. The description of STTs can be found in [3]. The user can call some additional functions during state transitions as described in [3].

The *dmonuser_help*() routine is called if **-h** is used.

2.4 Test Data Generator

A test data generator has been written and included in the online binary directories. This routine, called *rodemo* generates events with a fixed number of FED sub-blocks at regular intervals (every 1 s unless specified otherwise) and deposits the generated event in a named EMG partition. The *rodemo* command syntax is shown below.

```
rodemo [-h] [-c] [-s<num>] [-e<num>] [-f<num>] [-w<num>]
       [-d<nnn>] [-G<part>] [-t<nnn>] [-u<nnn>] [<rpm-options>]

   -h         prints the help message;
   -c         run continuously. Otherwise it will wait for begin run
              and end run messages;
   -s<num>    number of segments in the partition (8);
   -e<num>    number of events in a segment (1);
   -f<num>    number of FED's blocks in the events (2);
   -w<num>    number of words (4 bytes) in a FED block (32);
   -d<nnn>    nnn is the detector number in decimal;
   -G<part>   part is the event manager partition name;
   -t<nnn>    nnn is a number of seconds between generated events;
   -u<nnn>    nnn is a number of microseconds between events.
```

The **-t** and **-u** arguments are mutually exclusive. If neither **-t** nor **-u** is used the maximum event generation rate will be used. By default, *rodemo* produces events with a single SLB block containing 2 FED sub-blocks of 128 bytes each. The default version of *rodemo* should be available in the online library directories of all Hera-B online platforms. If the binary file is missing, or if a modified version of *rodemo* is needed, it can be rebuilt from the sources found in the online cvs repository under "evb/dem1996/rodemo.c" and ".../rdo.c". The make file is in "evb/makefile" in the online repository. The online installation guide [2] describes the environment variables used by this make file. The number of bytes and number of FEDs can be changed with parameters in the rdo.c source.

3 Example Program

```
/*
 *    Bit count monitor for use in finding quick
 *    occupancy numbers in the muon data.
 *
 *    The return status from dmonuser_loop() is being changed
 *    depending on the run state.
 *
 *    The following routines are included:
 *      dmonuser_help     user's program help
 *      dmonuser_init     program initialization
 *      dmonuser_begin    begin-run processing
 *      dmonuser_event    event processing
 *      dmonuser_loop     user's loop
 *      dmonuser_end      end-run processing
 *      dmonuser_halt     program termination
 */
#include <math.h>
#include <stdio.h>

#include "component.h"
#include "grr.h"
#include "etb.h"
#include "dmon.h"
#include "event.h"
static float count[32], sigma[32];
static int nevent = 0;


enum callmode user_loop_rc = never; /* user loop return code */
```

```
/*--------------------------------------------------------------------------*
 *                                                                          *
 * dmonuser_help()  user's help invoked if '-h' online parameter is present *
 *                                                                          *
 *--------------------------------------------------------------------------*/

int dmonuser_help(int argc , const char**argv) {

  printf("dmonuser_help()\n");
  return GRR_OK;
}

/*
 *     dmonuser_begin()  monitor program begin run routine
 */
int dmonuser_begin() {
user_loop_rc = onevent;
  return GRR_OK;}

/*
 *     dmonuser_end()    monitor program end run routine
 */
int dmonuser_end() {
user_loop_rc = always;
  return GRR_OK;}

/*
 *     dmonuser_event()   monitor program event processing routine
 */
int dmonuser_event(EVENT_HDR *event, int length) {
  SLB_SUBHDR *fedptr;
  int i, nbit, *iptr;
  unsigned int wd;
  float av, err;

/*-----------------------------------  Get a pointer to the
                                       first muon FED block */

/* not more in the new ETB  !!!!!!!!!!!!!!!!!!!!!!!           */
  fedptr = etb_get_fed_compid(event, 0, COMP_ID_MUON);
/* !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!           */
```

```
  if (!fedptr++) return GRR_OK;

  iptr = (int *) fedptr;

  for (i=0 ; i<32 ; i++)
    {
      nbit = 0;
      for (wd = *iptr++ ; wd ; wd = (wd >> 1) & 0x7fffffff)
        if (wd & 1) nbit++;
      count[i] += nbit;
      sigma[i] += nbit*nbit;
    }
  nevent++;
  if (nevent > 10000)
    {
      for (i=0 ; i<32 ; i++) {
        av = count[i]/nevent;
        err = sigma[i]/nevent - av*av;
        err = sqrt(err);
        printf("word %i average %f +/- %f \n", i, av, err);
        count[i] = 0;
        sigma[i] = 0;}
      nevent = 0;
    }

  return GRR_OK;} /* can be changed depending on the event processing
                     results. In this case the counter for errors
                     will be incremented */

/*
 *     dmonuser_loop()   monitor program idle loop routine
 */


int dmonuser_loop() {
time_t t;
static char my_time[32];
static int count = 0;

 par_update("Event");
```

```
 t = time(NULL);
 memset(my_time, 0, sizeof(my_time));
 strcpy(my_time, ctime(&t));
 my_time[strlen(my_time)-1] = 0;
 printf("dmonuser_loop(%d): %s\n", count++, my_time);
 fflush(0);
 switch(user_loop_rc) {
 case always:
   sleep(1);
 case onevent:
 case never:
 }
 return user_loop_rc;
}

/*
 *     dmonuser_halt()   terminate a monitor program
 */
int dmonuser_halt() {
  return GRR_OK;}

/*
 *     dmonuser_init()   initialize a monitor program
 */
int dmonuser_init() {
  grr_mask(COMP_M_DAQ);
  return GRR_OK;}
```

4 Hooks - User Provided Routines

The following user defined routines are called from the frame to allow the application designer to implement the specific functionality of the program. A template containing dummy versions of each of these routines is available in /hb/daq/Online_Library/examples/dmon-user.c.

Begin run routine.

Synopsis

```
int dmonuser_begin(void);
```

Parameters

none

Return values

GRR_OK processing completed.

Discussion

This routine is called at the start of processing each run. If continuous processing was requested (by the **-C** flag) *dmonuser_begin*() is called once when the program is first started.

dmonuser_end dmonuser_end

end run processing.

Synopsis

```
int dmonuser_end(void);
```

Parameters

none.

Return values

GRR_OK	processing completed.

Discussion

This routine is called at the end of each run. If continuous processing was requested (by the **-C** flag) *dmonuser_end*() is called once when the program is terminated.

Process one event.

Synopsis

```
int dmonuser_event(EVENT_HDR *event, int len);
```

Parameters

event buffer containing the event to be analyzed.

len length of the event data block.

Return values

GRR_OK processing completed (case 1).

not GRR_OK processing completed (case 2).

Discussion

dmonuser_event() is called once per event. The event is pointed to by *event* and the length of the block is passed in *len*. The ETB and ESP may be used for event processing. For the simplest statistics there are three internal counters those are to be incremented on the completion of event processing: *nevent* - case 1 , *nerror* - case 2. The whole number of events is stored in *ntrig* (see table 1).

dmonuser_halt dmonuser_halt

User termination routine.

Synopsis

```
int dmonuser_halt(void);
```

Parameters

none

Return values

GRR_OK completed.

Discussion

dmonuser_halt() is called once, just before the process terminates.

dmonuser_help dmonuser_help

User help routine.

Synopsis

```
int dmonuser_help(int argc, const char **argv);
```

Parameters

argc argument count from command line, if available

argv command line parameters, if available

Return values

GRR_OK help completed

Discussion

dmonuser_help() is called if **[-h]** option is used. The program terminates after returning from this function.

Set up user monitoring functions.

Synopsis

```
int dmonuser_init(int argc, const char **argv);
```

Parameters

argc argument count from command line, if available

argv command line parameters, if available

Return values

GRR_OK initialization completed.

GRR_INVALID environment is totally bogus

Discussion

At the time that *dmonuser_init*() is called, the frame has already performed the initialization of all permanent packages **rpm**,**emg**, **smc**, **par** although the process has not registered as a consumer. The user may register in the initialization routine if he desires. The initialization routine does all initialization needed for the functioning of the other *dmonuser* routines, and online packages they may require.

5 Online Frame Interface Routines

The user routines may control the function of the frame program in two ways. They are:

1. **Access routines** Functions are defined to allow the user routines to modify the frame execution mode and to give the user access to some frame facilities. These are described in the above pages.

2. **External variables** allow the user routines to control the online frame directly and to provide run-time parameters. These are summarized in table 1 The last 5 variables are set to the corresponding run parameters described in [4].

Variable	Meaning
active	flag which when set to zero terminates the online frame.
dmon_detect	detector ID (initialized by -D command line flag)
dmon_tmask	trigger mask (initialized by -T command line flag)
ntrig	the whole number of events consumed by the frame
nvent	the number of events processed by the user with the returned status **GRR_OK**
nerror	the number of events processed by the user with the returned status different from **GRR_OK**
ExpNumber	the experiment number
RunNumber	the run number
RunCfgName	the run configuration name
RunType	the run type
TrigLevel	the run triggers level
LogType	the run logging type

Table 1: External variables available to data monitor user routines.

3. **Published process parameters** allow the user to control and monitor the online frame by means of **par** package. These are summarized in table 2.

Name	Variable	Type	Access	Meaning
ExpNumber	ExpNumber	int	Read Only	the experiment number distributed by the central run control
RunNumber	RunNumber	int	Read Only	the run number distributed by the central run control
RunCfgName	RunCfgName	int	Read Only	the run configuration name distributed by the central run control
RunType	RunType	int	Read Only	the run type distributed by the central run control
TrigLevel	TrigLevel	int	Read Only	the run triggers level distributed by the central run control
LogType	LogType	int	Read Only	the run logging type distributed by the central run control
Active	active	int	RW	flag which when set to zero terminates the online frame.
Component	dmon_detect	int	Read Only	detector ID (initialized by -D command line flag)
Trigger	dmon_tmask	int	Read Only	trigger mask (initialized by -T command line flag)
Triggers	ntrig	int	RW	the whole number of events consumed by the frame
Events	nevent	int	RW	the number of events processed by the user with the returned status **GRR_OK**
Errors	nerror	int	RW	the number of events processed by the user with the returned status different from **GRR_OK**

Table 2: The variables accessible over **par** package facilities.

grr_error grr_error

Report an error.

Synopsis

```
int grr_error(int severity, const char *message, int error);
```

Parameters

severity severity of detected error

message message indicating the error location

error system (**GRR**) error code.

Return values

GRR_OK completed.

Discussion

An error message is formatted and sent to the local (detector) error logger. Fatal and severe errors are also sent to the global (root) error logger. If a local error logger can't be found, the error is sent to the global error logger. If no error logger is available, the error message is printed on the standard output device.

grr_mask grr_mask

Specify a detector mask.

Synopsis

```
int grr_mask(int detmask);
```

Parameters

detmask	mask with one bit set for each associated detector

Return values

GRR_OK	completed.
GRR_INVALID	a zero mask was specified

Discussion

The mask specifies the detectors with which the current process is associated. This mask is used in selecting messages for specific displays.

References

[1] M. Medinnis, Proposal for a Hera-B Message Handler Interface, Hera-B DAQ Note.

[2] J. Zweizig, Guide to Hera-B Online Library Installation, Hera-B DAQ Note.

[3] V. Rybnikov, The State Machine Control Package for the Hera-B Run Control System, Hera-B DAQ Note.

[4] D. Ressing, Hera-B naming conventions, Hera-B DAQ Note.