Midas Reference Manual

2.0.0-1

Generated by Doxygen 1.3.9.1

Thu Mar 8 23:04:47 2007

6 Midas Page Documentation

6.1 MIDAS Analyzer

- The Midas Analyzer application is composed of a collection of files providing a framework in which the user can gain access to the online data during data acquisition or offline data through a replay of a stored data save-set.

- The Midas distribution contains 2 directories where predefined set of analyzer files and their corresponding working demo code are available. The internal functionality of both example is similar and differ only on the histogram tool used for the data representation. These analyzer set are specific to 2 major data analysis tools i.e: **ROOT**, **HBOOK:**
  - **examples/experiment**: Analyzer tailored towards **ROOT** analysis
  - **examples/hbookexpt**: Analyzer tailored towards **HBOOK** with **PAW**.

- The purpose of the demo analyzer is to demonstrate the analyzer structure and to provide the user a set of code "template" for further development. The demo will run online or offline following the information given further down. The analysis goal is to:
  1. Initialize the ODB with predefined (user specific) structure (experim.h).
  2. Allocate memory space for histogram definition (booking).
  3. Acquire data from the frontend (or data file).
  4. Process the incoming data bank event-by-event through user specific code (module).
  5. Generate computed quantitied banks (in module).
  6. Fill (increment) predefined histogram with data available within the user code.
  7. Produce a result file containing histogram results and computed data (if possible) for further replay through dedicated analysis tool (PAW, ROOT).

- The analyzer is structured with the following files:
  - experim.h
    - ODB experiment include file defining the ODB structure required by the analyzer.
  - analyzer.c: main user core code.
    - Defines the incoming bank structures

    * Defines the analyzer modules
    * Initialize the ODB structure requirements
    * Provides Begin_of_Run and End_of_Run functions with run info logging example.
  - adccalib.c, adcsum.c, scaler.c (Root example)
    * Three user analysis modules to where events from the demo frontend.c sends data to.
  - **Makefile**
    * Specific makefile for building the corresponding frontend and analyzer code. The frontend code is build against the **camacnul.c** driver providing a simulated data stream.

- **ROOT** histogram booking code (excerpt of experiment/adcsum.c)
  - Histogram under ROOT is supported from version 1.9.5. This provides a cleaner way to organize the histogram grouping. This functionality is implemented with the function open_subfolder() and close_subfolder(). Dedicated Macro is also now available for histogram booking.

```
INT adc_summing_init(void)
{
   /* book ADC sum histo */
   hAdcSum = H1_BOOK("ADCSUM", "ADC sum", 500, 0, 10000);

   /* book ADC average in separate subfolder */
   open_subfolder("Average");
   hAdcAvg = H1_BOOK("ADCAVG", "ADC average", 500, 0, 10000);
   close_subfolder();

   return SUCCESS;
}
```

- **HBOOK** histogram booking code (excerpt of hbookexpt/adccalib.c)

```
INT adc_calib_init(void)
{
   char name[256];
   int i;

   /* book CADC histos */
   for (i = 0; i < N_ADC; i++) {
      sprintf(name, "CADC%02d", i);
      HBOOK1(ADCCALIB_ID_BASE + i, name, ADC_N_BINS,
             (float) ADC_X_LOW, (float) ADC_X_HIGH, 0.f);
   }

   return SUCCESS;
}
```

- The build is also specific to the type of histogram package involved and requires the proper libraries to generate the executable. Each directory has its own **Makefile:**

- **ROOT** (examples/experiment)
  - The environment **$ROOTSYS** is expected to point to a valid ROOT installed path.
  - The analyzer build requires a Midas core analyzer object file which should be present in the standard midas/<os>/lib directory. In order to have this file (rmana.o), the ROOTSYS had to be valid at the time of the Midas build too (See HAVE_HBOOK).
- **HBOOK** (examples/hbookexpt)
  - The analyzer build requires a Midas core analyzer object file which should be present in the standatd midas/<os>/lib directory. This file (hmana.o) doesn't require any specific library.
  - The analyzer build requires also at that stage to have access to some of the cernlib library files (See HAVE_HBOOK).
- **Analyzer Lite**
  - In the case private histogramming or simple analyzed data storage is requested, ROOT and HBOOK can be disabled by undefining both HAVE_ROOT and HAVE_HBOOK during the build.
  - This Lite version does't require any reference to the external histogramming package. Removal of specific definition histogram statement, function call from all the demo code (analyzer.c, adccalib.c, adcsum.c) needs to be done for successful build.
  - This Lite version will have no option of saving computed data from within the system analyzer framework, therefore this operation has to be performed by the user in the user code (module).

The following MultiStage Concept section describes in more details the analyzer concept and specific of the operation of the demo.

6.1.1 MultiStage Concept

In order to make data analysis more flexible, a multi-stage concept has been chosen for the analyzer. A raw event is passed through several stages in the analyzer, where each stage has a specific task. The stages read part of the event, analyze it and can add the results of the analysis back to the event. Therefore each stage in the chain can read all results from previous stages. The first stages in the chain typically deal with data calibration (adccalib.c), while the last stages contain the code which produces "physical" (adcsum.c) results like particle energies etc. The multi stage concept allows collaborations of people to use standard modules for the calibration stages which ensures that all members deal with the identical calibrated data, while the last stages can be modified by individuals to look at different aspects of the data. The stage system makes use of the MIDAS bank system. Each stage can read existing banks from an event and add

more banks with calculated data. Following picture gives an example of an analyzer consisting of three stages where the first two stages make an ADC and a MWPC calibration, respectively. They add a "Calibrated ADC" bank and a "MWPC" bank which are used by the third stage which calculates angles between particles:

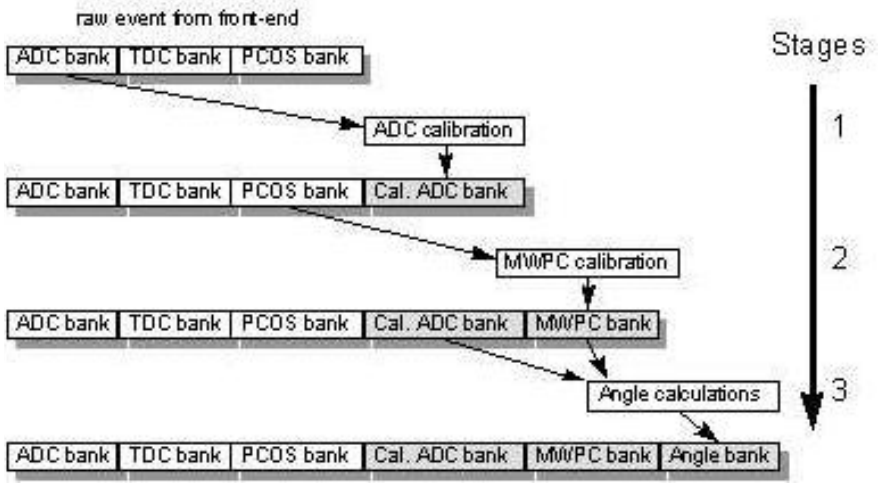


Figure 1: Three stage analyzer.

Since data is contained in MIDAS banks, the system knows how to interpret the data. By declaring new bank name in the analyzer.c as possible production data bank, a simple switch in the ODB gives the option to enable the recording of this bank into the result file. The user code for each stage is contained in a "module". Each module has a begin-of-run, end-of-run and an event routine. The BOR routine is typically used to book histograms, the EOR routine can do peak fitting etc. The event routine is called for each event that is received online or off-line.

6.1.1.1 Analyzer parameters Each analyzer has a dedicated directory in the ODB under which all the parameters realitve to this analyzer can be accessed. The path name is given from the "Analyzer name" specified in the analyzer.c under the analyzer_name. In case of concurrent analyzer, make sure that no conflict in name is present. By default the name is "Analyzer".

```
/* The analyzer name (client name) as seen by other MIDAS clients   */
char *analyzer_name = "Analyzer";
```

The ODB structure under it has the following fields

```
[host:expt:S]/Analyzer>ls -l
```

```
Key name                        Type    #Val  Size  Last Opn Mode Value
------------------------------------------------------------------------
Parameters                      DIR
Output                          DIR
Book N-tuples                   BOOL    1     4     1m   0   RWD  y
Bank switches                   DIR
Module switches                 DIR
ODB Load                        BOOL    1     4     19h  0   RWD  n
Trigger                         DIR
Scaler                          DIR
```

- **Parameters** : Created by the analyzer, contains all references to user parameters section.
- **Output** : System directory providing output control of the analyzer results.

```
[local:midas:S]/Analyzer>ls -lr output
Key name                        Type    #Val  Size  Last Opn Mode Value
------------------------------------------------------------------------
Output                          DIR
    Filename                    STRING  1     256   47h  0   RWD  run01100.root
    RWNT                        BOOL    1     4     47h  0   RWD  n
    Histo Dump                  BOOL    1     4     47h  0   RWD  n
    Histo Dump Filename         STRING  1     256   47h  0   RWD  his%05d.root
    Clear histos                BOOL    1     4     47h  0   RWD  y
    Last Histo Filename         STRING  1     256   47h  0   RWD  last.root
    Events to ODB               BOOL    1     4     47h  0   RWD  y
    Global Memory Name          STRING  1     8     47h  0   RWD  ONLN
```

  - **Filename** : Replay result file name.
  - **RWNT** : To be ignored for **ROOT**, N-Tuple Raw-wise data type.
  - **Histo Dump** : Enable the saving of the run results (see next field)
  - **Histo Dump Filename** : Online Result file name
  - **Clear Histos** : Boolean flag to enable the clearing of all histograms at the begining of each run (online or offline).
  - **Last Histo Filename** : Temporary results file for recovery procedure.
  - **Event to ODB** : Boolean flag for debugging purpose allowing a copy of the data to be sent to the ODB at regular time interval (1 second).
  - **Global Memory Name** : Shared memory name for communication between Midas and HBOOK. To be ignored for **ROOT** as the data sharing is done through a TCP/IP channel.

- **Bank switches** : Contains the list of all declared banks (BANK_LIST in analyzer.c) to be enabled for writing to the output result file. By default all the banks are disabled.

```
[local:midas:S]/Analyzer>ls "Bank switches" -l
Key name                        Type    #Val  Size  Last Opn Mode Value
```

```
--------------------------------------------------------------------------------
ADC0                            DWORD   1     4     1h    0    RWD  0
TDC0                            DWORD   1     4     1h    0    RWD  0
CADC                            DWORD   1     4     1h    0    RWD  0
ASUM                            DWORD   1     4     1h    0    RWD  0
SCLR                            DWORD   1     4     1h    0    RWD  0
ACUM                            DWORD   1     4     1h    0    RWD  0
```

- **Module switches** : Contains the list of all declared module (ANA_MODULE in analyzer.c) to be controlled (by default all modules are enabled)

```
[local:midas:S]/Analyzer>ls "module switches" -l
Key name                        Type    #Val  Size  Last Opn Mode Value
--------------------------------------------------------------------------------
ADC calibration                 BOOL    1     4     1h    0    RWD  y
ADC summing                     BOOL    1     4     1h    0    RWD  y
Scaler accumulation             BOOL    1     4     1h    0    RWD  y
```

- **ODB Load** : Boolean switch to allow retrieval of the entire ODB structure from the input data file. Used only during offline, this option permits to replay the data in the same exact condition as during online. All the ODB parameter settings will be restored to their last value as at the end of the data acquisition of this particular run.
- **Trigger**, **Scaler** : Subdirectories of all the declared requested event. (ANALYZE_REQUEST in analyzer.c)
- **BOOK N_tuples** : Boolean flag for booking N-Tuples at the initialization of the module. This flag is specific to the **HBOOK** analyzer.
- **BOOK TTree** : Boolean flag for booking TTree at the initialization of the module. This flag is specific to the **ROOT** analyzer.

6.1.1.2 Analyzer Module parameters

Each analyzer module can contain a set of parameters to either control its behavior, . These parameters are kept in the ODB under /Analyzer/Parameters/<module name> and mapped automatically to C structures in the analyzer modules. Changing these values in the ODB can therefore control the analyzer. In order to keep the ODB variables and the C structure definitions matched, the ODBEdit command **make** generates the file experim.h which contains C structures for all the analyzer parameters. This file is included in all analyzer source code files and provides access to the parameters from within the module file under the name <module name>_param.

- Module name: adc_calib_module (extern ANA_MODULE adc_calib_module from analyzer.c)
- Module file name: adccalib.c

- Module structure declaration in adccalib.c:

```
ANA_MODULE adc_calib_module = {
   "ADC calibration",           /* module name           */
   "Stefan Ritt",               /* author                */
   adc_calib,                   /* event routine         */
   adc_calib_bor,               /* BOR routine           */
   adc_calib_eor,               /* EOR routine           */
   adc_calib_init,              /* init routine          */
   NULL,                        /* exit routine          */
   &adccalib_param,             /* parameter structure   */
   sizeof(adccalib_param),      /* structure size        */
   adc_calibration_param_str,   /* initial parameters    */
};
```

  - ODB parameter variable in the code: <module name>_param -> adccalib_param (from adc_calib_module, the _ is dropped, module is removed)
  - ODB parameter path: /<Analyzer>/Parameters/ADC calibration/ (using the module name from the structure)
  - Access to the module parameter:

```
    /* subtract pedestal */
   for (i = 0; i < N_ADC; i++)
      cadc[i] = (float) ((double) pdata[i] - adccalib_param.pedestal[i] + 0.5);
```

  - ODB module parameter declaration

```
 [local:midas:S]Parameters>pwd
/Analyzer/Parameters
[local:midas:S]Parameters>ls -lr
Key name                      Type    #Val  Size  Last Opn Mode Value
---------------------------------------------------------------------------
Parameters                    DIR
    ADC calibration           DIR
        Pedestal              INT     8     4     47h  0   RWD
                                      [0]             174
                                      [1]             194
                                      [2]             176
                                      [3]             182
                                      [4]             185
                                      [5]             215
                                      [6]             202
                                      [7]             202
        Software Gain         FLOAT   8     4     47h  0   RWD
                                      [0]             1
                                      [1]             1
                                      [2]             1
                                      [3]             1
                                      [4]             1
                                      [5]             1
                                      [6]             1
                                      [7]             1
        Histo threshold       DOUBLE  1     8     47h  0   RWD  20
    ADC summing               DIR
```

```
    ADC threshold          FLOAT   1     4     47h  0   RWD  5
  Global                   DIR
    ADC Threshold          FLOAT   1     4     47h  0   RWD  5
```

6.1.1.3 Analyzer Flow chart The general operation of the analyzer can be summerized as follow:

- The analyzer is a Midas client at the same level as the odb or any other Midas Utilities application.
- When the analyzer is started with the proper argument (experiment, host for remote connection or -i input_file, -o output_file for off-line use), the initialization phase will setup the following items:

  1. Setup the internal list of defined module.

```
ANA_MODULE *trigger_module[] = {
   &adc_calib_module,
   &adc_summing_module,
   NULL
};
```

  2. Setup the internal list of banks.

```
BANK_LIST ana_trigger_bank_list[] = {

   /* online banks */
   {"ADC0", TID_STRUCT, sizeof(ADC0_BANK), ana_adc0_bank_str}
   ,
   {"TDC0", TID_WORD, N_TDC, NULL}
   , ...
```

  3. Define the internal event request structure and attaching the corresponding module and bank list.

```
ANALYZE_REQUEST analyze_request[] = {
{"Trigger",                  /* equipment name */
 {1,                         /* event ID */
  TRIGGER_ALL,               /* trigger mask */
  GET_SOME,                  /* get some events */
  "SYSTEM",                  /* event buffer */
  TRUE,                      /* enabled */
  "", "",}
 ,
 NULL,                       /* analyzer routine */
 trigger_module,             /* module list */
 ana_trigger_bank_list,      /* bank list */
 1000,                       /* RWNT buffer size */
 TRUE,                       /* Use tests for this event */
 }
, ...
```

4. Setup the ODB path for each defined module.
5. Book the defined histograms of each module.
6. Book memory for N-Tuples or TTree.
7. Initialize the internal "hotlinks" to the defined ODB analyzer module parameter path.
   - Once the analyzer is in idle state (for online only), it will wakeup on the transition "Begin-of-Run" and go sequencially through all the modules BOR functions. which generally will ensure proper histogramming booking and possible clearing. It will resume its idle state waiting for the arrival of an event matching one of the event request structure declared during initialization (analyzer.c)

– In case of off-line analysis, once the initialization phase successfully complete, it will go through the BOR and start the event-by-event acquisition.

```
INT analyzer_init()
{
  HNDLE hDB, hKey;
  char str[80];

  RUNINFO_STR(runinfo_str);
  EXP_PARAM_STR(exp_param_str);
  GLOBAL_PARAM_STR(global_param_str);
  TRIGGER_SETTINGS_STR(trigger_settings_str);

  /* open ODB structures */
  cm_get_experiment_database(&hDB, NULL);
  db_create_record(hDB, 0, "/Runinfo", strcomb(runinfo_str));
  db_find_key(hDB, 0, "/Runinfo", &hKey);
  if (db_open_record(hDB, hKey, &runinfo, sizeof(runinfo), MODE_READ, NULL, NULL) !=
    DB_SUCCESS) {
   cm_msg(MERROR, "analyzer_init", "Cannot open \"/Runinfo\" tree in ODB");
   return 0;
  }
```

1. When an event is received and matches one the the event request structure, it is passed in sequence to all the defined module for that event request (see in the ANALYZER_REQUEST structure the line containing the comment module list.
   - If some of the module don't need to be invoked by the incoming event, it can be disabled interactively through ODB from the /analyzer/Module switches directory

```
[ladd00:p3a:Stopped]Module switches>ls
ADC calibration                 y
ADC summing                     y
Scaler accumulation             y
[ladd00:p3a:Stopped]Module switches>
```

   - if the module switch is enabled, the event will be presented in the module at the defined event-by-event function declared in the module structure (adccalib.c) in this case the function is adc_calib().

– The Midas event header is accessible through the pointer **pheader** while the data is located by the pointer **pevent**

```
INT adc_calib(EVENT_HEADER * pheader, void *pevent)
{
 INT i;
 WORD *pdata;
 float *cadc;

 /* look for ADC0 bank, return if not present */
 if (!bk_locate(pevent, "ADC0", &pdata))
  return 1;
```

- Refer to the example found under **examples/experiment** directory for **ROOT** analyzer and **examples/hbookexpt** directory for **HBOOK** analyzer.

6.1.1.4 HBOOK analyzer description (old doc)

PAWC_DEFINE(8000000);

This defines a section of 8 megabytes or 2 megawords of share memory for HBOOK/Midas data storage. This definition is found in analyzer.c. In case many histograms are booked in the user code, this value probably has to be increased in order not to crash HBOOK. If the analyzer runs online, the section is kept in shared memory. In case the operating system only supports a smaller amount of shared memory, this value has to be decreased. Next, the file contains the analyzer name

char ∗analyzer_name = "Analyzer";

under which the analyzer appears in the ODB (via the ODBEdit command scl). This also determines the analyzer root tree name as /Analyzer. In case several analyzers are running simultaneously (in case of distributed analysis on different machines for example), they have to use different names like Analyzer1 and Analyzer2 which then creates two separate ODB trees /Analyzer1 and /Analyzer2 which is necessary to control the analyzers individually. Following structures are then defined in analyzer.c: runinfo, global_param, exp_param and trigger_settings. They correspond to the ODB trees /Runinfo, /Analyzer/Parameters/Global, /Experiment/Run parameters and /Equipment/Trigger/Settings, respectively. The mapping is done in the analyzer_init() routine. Any analyzer module (via an extern statement) can use the contents of these structures. If the experiment parameters contain an flag to indicate the run type for example, the analyzer can analyze calibration and data runs differently. The module declaration section in analyzer.c defines two "chains" of modules, one for trigger events and one for scaler events. The framework calls these according to their order in these lists. The modules of type ANA_MODULE are defined in their source code file. The enabled flag for each module is copied to the ODB under /Analyzer/Module switches. By setting this flag zero in the ODB, modules can be disabled temporarily. Next, all banks have to be defined. This is necessary because the framework automatically books N-tuples for all banks at startup before any event is received. Online banks which come from the frontend are first defined, then banks created by the analyzer:

```
...
// online banks
{ "ADC0", TID_DWORD, N_ADC, NULL },
{ "TDC0", TID_DWORD, N_TDC, NULL },

// calculated banks
{ "CADC", TID_FLOAT, N_ADC, NULL },
{ "ASUM", TID_STRUCT, sizeof(ASUM_BANK),
 asum_bank_str },
```

The first entry is the bank name, the second the bank type. The type has to match the type which is created by the frontend. The type TID_STRUCT is a special bank type. These banks have a fixed length which matches a C structure. This is useful when an analyzer wants to access named variables inside a bank like asum_bank.sum. The third entry is the size of the bank in bytes in case of structured banks or the maximum number of items (not bytes!) in case of variable length banks. The last entry is the ASCII representation of the bank in case of structured banks. This is used to create the bank on startup under /Equipment/Trigger/Variables/<bank name>.

The next section in analyzer.c defines the ANALYZE_REQUEST list. This determines which events are received and which routines are called to analyze these events. A request can either contain an "analyzer routine" which is called to analyze the event or a "module list" which has been defined above. In the latter case all modules are called for each event. The requests are copied to the ODB under /Analyzer/<equipment name>/Common. Statistics like number of analyzed events is written under /Analyzer/<equipment name>/Statistics. This scheme is very similar to the frontend Common and Statistics tree under /Equipment/<equipment name>/. The last entry of the analyzer request determines the HBOOK buffer size for online N-tuples. The analyzer_init() and analyzer_exit() routines are called when the analyzer starts or exits, while the ana_begin_of_run() and ana_end_of_run() are called at the beginning and end of each run. The ana_end_of_run() routine in the example code writes a run log file runlog.txt which contains the current time, run number, run start time and number of received events.

If more parameters are necessary, perform the following procedure:

1. modify/add new parameters in the current ODB.

```
[host:expt:S]ADC calibration>set Pedestal[9] 3
[host:expt:S]ADC calibration>set "Software Gain[9]" 3
[host:expt:S]ADC calibration>create double "Upper threshold"
[host:expt:S]ADC calibration>set "Upper threshold" 400
[host:expt:S]ADC calibration>ls -lr
Key name                        Type    #Val  Size  Last Opn Mode Value
---------------------------------------------------------------------------
ADC calibration                 DIR
    Pedestal                    INT     10    4     2m   0   RWD
                                        [0]             174
                                        [1]             194
                                        [2]             176
                                        [3]             182
```

```
                                        [4]              185
                                        [5]              215
                                        [6]              202
                                        [7]              202
                                        [8]              0
                                        [9]              3
    Software Gain               FLOAT   10    4     2m   0    RWD
                                        [0]              1
                                        [1]              1
                                        [2]              1
                                        [3]              1
                                        [4]              1
                                        [5]              1
                                        [6]              1
                                        [7]              1
                                        [8]              0
                                        [9]              0
    Histo threshold             DOUBLE  1     8     53m  0    RWD  20
    Upper threshold             DOUBLE  1     4     3s   0    RWD  400
```

2. Generate experim.h

```
[host:expt:S]ADC calibration>make
"experim.h" has been written to /home/midas/online
```

3. Update the module with the new parameters.

```
---> adccalib.c
...
fill ADC histos if above threshold
for (i=0 ; i<n_adc ; i++)
if ((cadc[i] > (float) adccalib_param.histo_threshold)
 && (cadc[i] < (float) adccalib_param.upper_threshold))
    HF1(ADCCALIB_ID_BASE+i, cadc[i], 1.f);
```

4. Rebuild the analyzer.

In the case global parameter is necessary for several modules, start by doing the step 1 & 2 from the enumeration above and carry on with the following procedure below:

1. Declare the parameter global in analyzer.c

```
// ODB structures
...
GLOBAL_PARAM     global_param;
...
```

2. Update ODB structure and open record for that parameter (hot link).

```
---> analyzer.c
...
sprintf(str, "/%s/Parameters/Global", analyzer_name);
```

```
db_create_record(hDB, 0, str, strcomb(global_param_str));
db_find_key(hDB, 0, str, &hKey);
if (db_open_record(hDB, hKey, &global_param
    , sizeof(global_param), MODE_READ, NULL, NULL) != DB_SUCCESS) {
  cm_msg(MERROR, "analyzer_init", "Cannot open \"%s\" tree in ODB", str);
  return 0;
}
```

3. Declare the parameter **extern** in the required module

```
---> adccalib.c
...
extern GLOBAL_PARAM  global_param;
...
```

6.1.1.5 Online usage with PAW Once the analyzer is build, run it by entering: **analyzer [-h <host name>] [-e <exp name>]**

where <host name> and <exp name> are optional parameters to connect the analyzer to a remote back-end computer. This attaches the analyzer to the ODB, initializes all modules, creates the PAW shared memory and starts receiving events from the system buffer. Then start PAW and connect to the shared memory and display its contents

```
PAW > global_s onln
PAW > hist/list
    1  Trigger
    2  Scaler
 1000  CADC00
 1001  CADC01
 1002  CADC02
 1003  CADC03
 1004  CADC04
 1005  CADC05
 1006  CADC06
 1007  CADC07
 2000  ADC sum
```

For each equipment, a N-tuple is created with a N-tuple ID equal to the event ID. The CADC histograms are created from the adc_calib_bor() routine in adccalib.c. The N-tuple contents is derived from the banks of the trigger event. Each bank has a switch under /Analyzer/Bank switches. If the switch is on (1), the bank is contained in the N-tuple. The switches can be modified during runtime causing the N-tuples to be rebooked. The N-tuples can be plotted with the standard PAW commands:

```
PAW > nt/print 1
...
PAW > nt/plot 1.sum
PAW > nt/plot 1.sum cadc0>3000
```

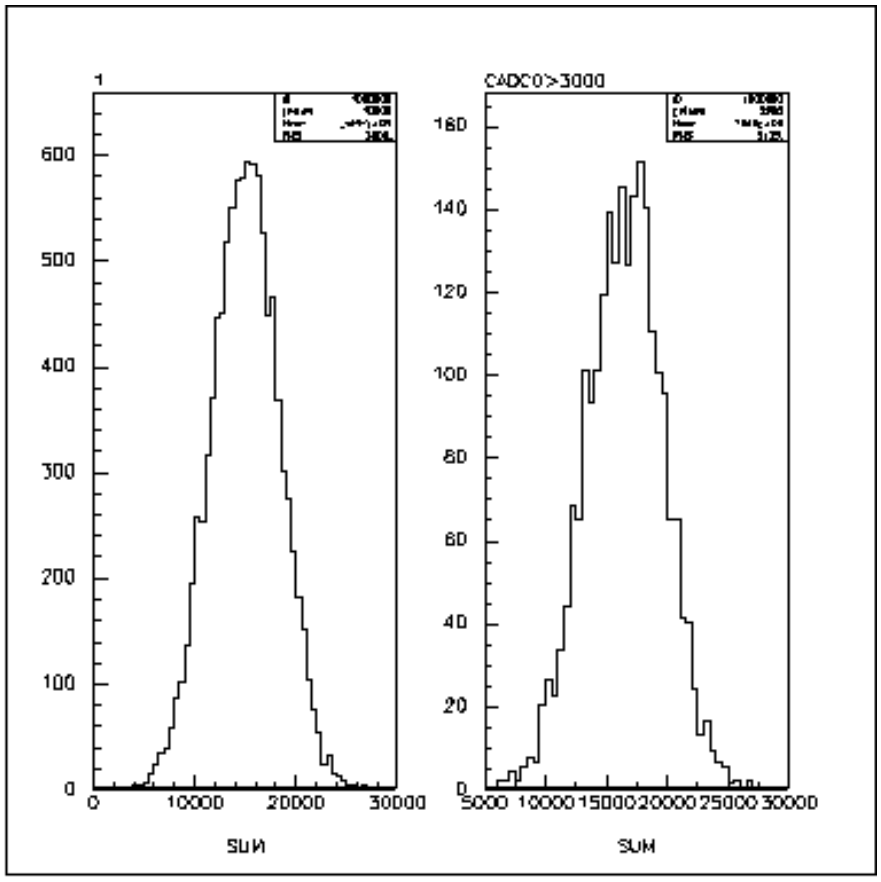


Figure 2: PAW output for online N-tuples.

While histograms contain the full statistics of a run, N-tuples are kept in a ring-buffer. The size of this buffer is defined in the ANALYZE_REQUEST structure as the last parameter. A value of 10000 creates a buffer which contains N-tuples for 10000 events. After 10000 events, the first events are overwritten. If the value is increased, it might be that the PAWC size (PAWC_DEFINE in analyzer.c) has to be increased, too. An advantage of keeping the last 10000 events in a buffer is that cuts can be made immediately without having to wait for histograms to be filled. On the other hand care has to be taken in interpreting the data. If modifications in the hardware are made during a run, events which reflect the modifications are mixed with old data. To clear the ring-buffer for a N-tuple or a histogram during a run, the ODBEdit command **[local]/>hi analyzer <id>**

where <id> is the N-tuple ID or histogram ID. An ID of zero clears all histograms but no N-tuples. The analyzer has two more ODB switches of interest when running on-

line. The /Analyzer/Output/Histo Dump flag and /Analyzer/Output/Histo Dump Filename determine if HBOOK histograms are written after a run. This file contains all histograms and the last ring-buffer of N-tuples. It can be read in with PAW:

```
PAW >hi/file 1 run00001.rz 8190
PAW > ldir
```

The /Analyzer/Output/Clear histos flag tells the analyzer to clear all histograms and N-tuples at the beginning of a run. If turned off, histograms can be accumulated over several runs.

6.1.1.6 Offline usage with PAW The analyzer can be used for off-line analysis without recompilation. It can read from MIDAS binary files (∗.mid), analyze the data the same way as online, and the write the result to an output file in MIDAS binary format, ASCII format or HBOOK RZ format. If written to a RZ file, the output contains all histograms and N-tuples as online, with the difference that the N-tuples contain all events, not only the last 10000. The contents of the N-tuples can be a combination of raw event data and calculated data. Banks can be turned on and off in the output via the /Analyzer/Bank switches flags. Individual modules can be activated/deactivated via the /Analyzer/Module switches flags.

The RZ files can be analyzed and plotted with PAW. Following flags are available when the analyzer is started off-line:

- -i [filename1] [filename2] ... Input file name(s). Up to ten different file names can be specified in a -i statement. File names can contain the sequence "%05d" which is replaced with the current run number in conjunction with the -r flag. Following filename extensions are recognized by the analyzer: .mid (MIDAS binary), .asc (ASCII data), .mid.gz (MIDAS binary gnu-zipped) and .asc.gz (ASCII data gnu-zipped). Files are un-zipped on-the-fly.
- -o [filename] Output file name. The file names can contain the sequence "%05d" which is replaced with the current run number in conjunction with the -r flag. Following file formats can be generated: .mid (MIDAS binary), .asc (ASCII data), .rz (HBOOK RZ file), .mid.gz (MIDAS binary gnu-zipped) and .asc.gz (ASCII data gnu-zipped). For HBOOK files, CWNT are used by default. RWNT can be produced by specifying the -w flag. Files are zipped on-the-fly.
- -r [range] Range of run numbers to be analyzed like -r 120 125 to analyze runs 120 to 125 (inclusive). The -r flag must be used with a "%05d" in the input file name.
- -n [count] Analyze only count events. Since the number of events for all event types is considered, one might get less than count trigger events if some scaler or other events are present in the data.
- -n [first] [last] Analyze only events with serial numbers between first and last.

- -n [first] [last] [n] Analyze every n-th event from first to last.
- -c [filename1] [filename2] ... Load configuration file name(s) before analyzing a run. File names may contain a "%05d" to be replaced with the run number. If more than one file is specified, parameters from the first file get superseded from the second file and so on. Parameters are stored in the ODB and can be read by the analyzer modules. They are conserved even after the analyzer has stopped. Therefore, only parameters which change between runs have to be loaded every time. To set a parameter like /Analyzer/Parameters/ADC summing/offset one would load a configuration file which contains:

  ```
  [Analyzer/Parameters/ADC summing]
  Offset = FLOAT : 123
  ```

  Loaded parameters can be inspected with ODBEdit after the analyzer has been started.
- -p [param=value] Set individual parameters to a specific value. Overrides any setting in configuration files. Parameter names are relative to the /Analyzer/Parameters directory. To set the key /Analyzer/Parameters/ADC summing/offset to a specific value, one uses -p "ADC summing/offset"=123. The quotation marks are necessary since the key name contains a blank. To specify a parameter which is not under the /Analyzer/Parameters tree, one uses the full path (including the initial "/") of the parameter like -p "/Experiment/Run Parameters/Run mode"=1.
- -w Produce row-wise N-tuples in output RZ file. By default, column-wise N-tuples are used.
- -v Convert only input file to output file. Useful for format conversions. No data analysis is performed.
- -d Debug flag when started the analyzer from a debugger. Prevents the system to kill the analyzer when the debugger stops at a breakpoint.

6.2 Data format

Utilities - Top - Supported hardware

Midas supports two differents data format so far. A possible new candidate would be the NeXus format, but presently no implementation has been developed.

- Midas format
- YBOS format

6.2.1 Midas format

Special formats are used in MIDAS for the event header, banks and when writing to disk or tape. This appendix explains these formats in detail. Each event carries a 16-byte header. The header is generated by the front-end with the bm_compose_event() routine and is used by consumers to distinguish between different events. The header is defined in the EVENT_HEADER structure in midas.h. It has following structure:

Event and bank headers with data block.

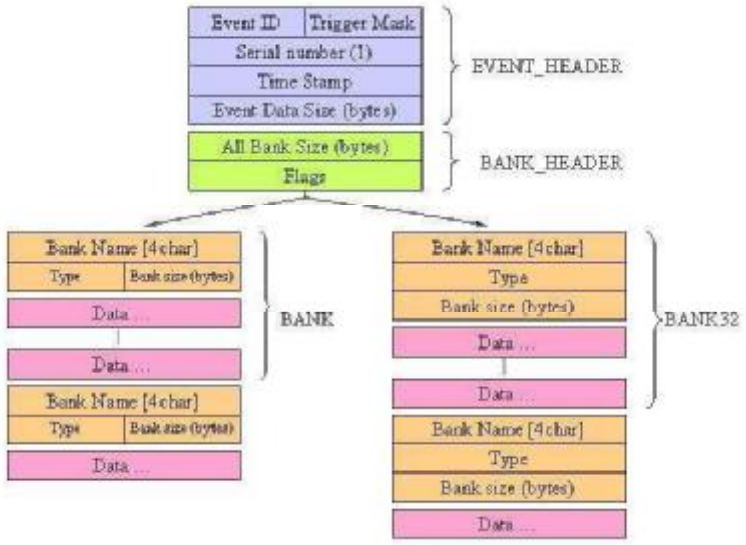


Figure 3: Event and bank headers with data block.

The event ID describes the type of event. Usually 1 is used for triggered events, 2 for scaler events, 3 for HV events etc. The trigger mask can be used to describe the sub-type of an event. A trigger event can have different trigger sources like "physics event", "calibration event", "clock event". These trigger sources are usually read in by the front-end in a pattern unit. Consumers can request events with a specific triggering mask. The serial number starts at one and is incremented by the front-end for each event. The time stamp is written by the front-end before an event is read out. It uses the time() function which returns the time in seconds since 1.1.1970 00:00:00 UTC. The data size contains the number of bytes that follows the event header. The data area of the event can contain information in any user format, although only certain formats

are supported when events are copied to the ODB or written by the logger in ASCII format. Event headers are always kept in the byte ordering of the local machine. If events are sent over the network between computers with different byte ordering, the event header is swapped automatically, but not the event contents.

- [Bank Format] Events in MIDAS format contain "MIDAS banks". A bank is a substructure of an event and can contain only one type of data, either a single value or an array of values. Banks have a name of exactly four characters, which are treated, as a bank ID. Banks in an event consist of a global bank header and an individual bank header for each bank. Following picture shows a MIDAS event containing banks:

  The "data size total" is the size in bytes of all bank headers and bank data. Flags are currently not used. The bank header contains four characters as identification, a bank type that is one of the TID_xxx values defined in midas.h, and the data size in bytes. If the byte ordering of the contents of a complete event has to be swapped, the routine bk_swap() can be used.

- [Tape Format] Events are written to disk files without any reformatting. For tapes, a fixed block size is used. The block size TAPE_BUFFER_SIZE is defined in midas.h and usually 32kB. Three special events are produced by the system. A begin-of-run (BOR) and end-of-run (EOR) event is produced which contains an ASCII dump of the ODB in its data area. Their IDs is 0x8000 (BOR) and 0x8001 (EOR). A message event (ID 0x8002) is created if Log messages is enabled in the logger channel setting. The message is contained in the data area as an ASCII string. The BOR event has the number MIDAS_MAGIC (0x494d or 'MI') as the trigger mask and the current run number as the serial number. A tape can therefore be identified as a MIDAS formatted tape. The routine tape_copy() in the utility mtape.c is an example of how to read a tape in MIDAS format.

6.2.2 YBOS format

As mentioned earlier the YBOS documentation is available at the following URL address: `Ybos site` Originally YBOS is a collection of FORTRAN functions which facilitate the manipulation of group of data. It also describes a mode of encoding/storing data in an organized way. YBOS defines specific ways for:

- Gathering related data (bank structure).
- Gathering banks structure (logical record).
- Gathering/Writing/Reading logical record from/to storage device such as disk or tape. (Physical record).

YBOS is organized on a 4-byte alignment structure.

The YBOS library function provides all the tools for manipulation of the above mentioned elements in a independent Operating System like. But the implementation of the YBOS part in Midas does not use any reference to the YBOS library code. Instead only the strict necessary functions have to be re-written in C and incorporated into the Midas package. This has been motivated by the fact that only a sub-set of function is essential to the operation of:

- The front-end code: for the composition of the YBOS event (bank structure, logical record).
- The data logger: for writing data to storage device (physical record).

This Midas/YBOS implementation restricts the user to a subset of the YBOS package only for the front-end part. It doesn't prevent him/her to use the full YBOS library for stand alone program accessing data file written by Midas.

The YBOS implementation under Midas has the following restrictions:

- Single leveled bank structures only (no recursive bank allowed).
- Bank structure of the following type: ASCII, BINARY, WORD, DOUBLE WORD, IEEE FLOATING.
- No mixed data type bank structure allowed.
- Logical Record format (Event Format) In the YBOS terminology a logical record refers to a collection of YBOS bank while in the Midas front-end, it can be referred to as an event. The logical record consists of a logical record length of a 32bit-word size followed by a single or collection of YBOS bank. The logical record length counts the number of double word (32bit word) composing the record without counting itself.

YBOS uses "double word" unit for all length references.

- [Bank Format] The YBOS bank is composed of a bank header 5 double long words followed by the data section which has to end on a 4 bytes boundary.

Ybos Event and bank headers with data block.

Figure 4: Ybos Event and bank headers with data block.

The bank length parameter corresponds to the size of the data section in double word count + 1. The supported bank type are defined in the ybos.h file see YBOS Bank Types.

- [Physical Record (Tape/Disk Format)] The YBOS physical record structure is based on a fixed block size (8190 double words) composed of a physical record header followed by data from logical records.

Ybos Physical record structure with data block.

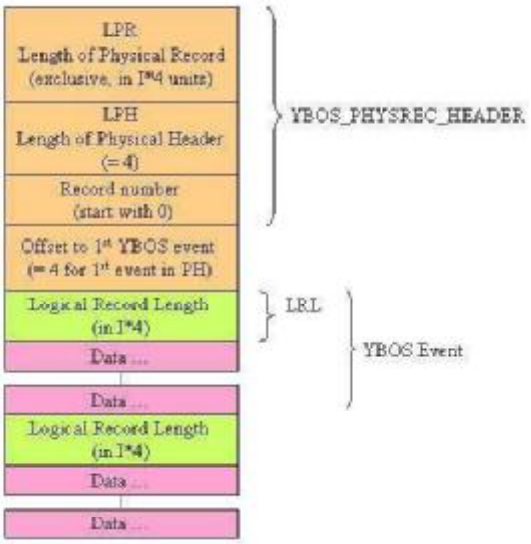


Figure 5: Ybos Physical record structure with data block..

The Offset is computed with the following rules:

- If the logical record fits completely in the space of the physical record, the offset value in the physical record header will be 4.
- If the block contains first the left over fragment of the previous event started in the previous block, the offset will be equal to the length of the physical record header + the left over fragment size.
- If the logical record extent beyond a full block, the offset will be set to -1.
- The mark of the end of file is defined with a logical record length set to -1.

Utilities - Top - Supported hardware

6.3 Supported hardware

Data format - Top - CAMAC and VME access function call

The driver library is continuously extended to suit the needs of various experiments based on the selected hardware modules. Not all commercially available modules are

included as we don't have all the modules in hand. But you're more than welcome to contribute by providing your driver code if the module that you're using is not yet listed.

The **/drivers** directory is subdivided in several directories which refers to either the type of bus ie: CAMAC, FastBus, VME, PCI, USB or type of software layer such as Class, Device, Bus.

The software layers sections are used in particular for Slow Control System. Example are available in the distribution under **examples/slowcont/frontend**.c including the **hv** and **multi** class with the **nulldev** device and **null** bus driver. Note: not all the device drivers implement the triple layer (Class,Device,Bus) as some include directly the hardware calls in the device layer. Please contact midas for specific support or for submitting new drivers.

Non exhaustive Drivers/ directory structure

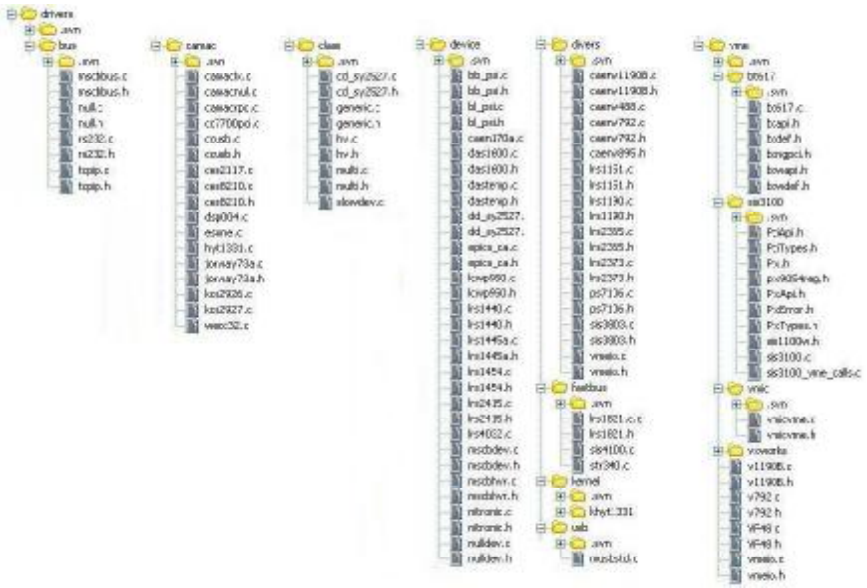


Figure 6: Drivers/ directory structure

- CAMAC drivers This section is slowly getting obsolete. But still some ISA and PCI interface are in use. Most recent development is the USB/CAMAC interface from Wiener (CCUSB). While this interface permits **CAMAC Command Stacks** this option is not yet supported by the Midas API limiting the access speed of a R/W 24bit cycle to ∼360us!

- VME drivers The VME API has been revisited for a better function call set. Not all the hardware modules have been ported to this new scheme. DMA and Interrupt support have been included. The main hardware support is for the SBS PCI/VME, SIS PCI/VME, VMIC processor.

- USB drivers USB is getting popular in particular for the MSCB system. Following the same concept as for the CAMAC and VME, the **musbstd.h/c** is available for USB access.

- GPIB drivers

- Other drivers This include the TCP/IP, Serial access layer.

6.3.1 CAMAC drivers

The CAMAC drivers can be used in different configuration and may have special behaviors depending on the type of hardware involved. Below are summurized some remarks about these particular hardware modules.

- CAMAC controllers
  - [hyt1331.c] This interface uses an ISA board to connect to the crate controller. This card implement a "fast" readout cycle by re-triggering the CAMAC read at the end of the previous one. This feature is unfortunately not reliable when fast processor is used. Wrong returned data can be expected when CPU clocks is above 250MHz. Attempt on "slowing down" the IO through software has not guaranteed perfect result. Contact has been taken with HYTEC in order to see if possible fix can be applied to the interface. First revision of the PC-card PAL has been tested but did not show improvement. CVS version of the hyt1331.c until 1.2 contains "fast readout cycle" and should not be trusted. CVS 1.3 driver revision contains a patch to this problem. In the mean time you can apply your own patch (see Frequently Asked Questions) and also Hytec)
  - **[hyt1331.c Version >= 1.8.3]** This version has been modified for 5331 PCI card support running under the dio task.
  - **[khyt1331.c Version >= 1.8.3]** A full Linux driver is available for the 5331 PCI card interfacing to the hyt1331. The kernel driver has been written for the Linux kernel 2.4.2, which comes with RedHat 7.1. It could be ported back to the 2.2.x kernel because no special feature of 2.4.x are used, although many data structures and function parameters have changed between 2.2 and 2.4, which makes the porting a bit painful. The driver supports only one 5331 card with up to four CAMAC crates.

- **[kcs292x.c]** The 2926 is an 8 bit ISA board, while the 2927 is a 16bit ISA board. An equivalent PCI interface (2915) exists but is not yet supported by Midas (See KCS). No support for Windowx yet.

  Both cards can be used also through a proper Linux driver *camaclx.c*. This requires to first load a module *camac-kcs292x*.o. This software is available but not part of the Midas distribution yet. Please contact midas for further information.
- **[wecc32.c]** The CAMAC crate controller CC32 interface to a PCI card... you will need the proper Linux module... Currently under test. Windows-NT and W95 drivers available but not implemented under Midas. (see CC32)
- **[dsp004.c]** The dsp004 is an 8 bit ISA board PC interface which connect to the PC6002 CAMAC crate controller. This module is not being manufactured anymore, but somehow several labs still have that controller in use.
- **[ces8210.c]** The CAMAC crate controller CBD8210 interface is a VME module to give access up to 7 CAMAC crate. In conjunction with the mvmestd.h and mcstd.h, this driver can be used on any Midas/VME interface.
- **[jorway73a.c]** The CAMAC crate controller Jorway73a is accessed through SCSI commands. This driver implement the mcstd.h calls.

- CAMAC drivers
  - [camacnul.c] Handy fake CAMAC driver for code development.
  - [camacrpc.c] Remote Procedure Call CAMAC driver used for accessing the CAMAC server part of the standard Midas frontend code. This driver is used for example in the mcnaf task, mhttpd task utilities.

6.3.2 VME drivers

The VME modules drivers can be interfaced to any type of PCI/VME controller. This is done by dedicated Midas VME Standard calls from the mvmestd.h files.

- PCI/VME interface
  - [sis1100.c] PCI/VME with optical fiber link. Driver is under development (March 2002). (see SIS).
  - [bt617.c] Routines for accessing VME over SBS Bit3 Model 617 interface under Windows NT using the NT device driver Model 983 and under Linux using the vmehb device driver. The VME calls are implemented for the "mvmestd" Midas VME Standard. (see Bit3).

  - [wevmemm.c] PCI/VME Wiener board supported. (see Wiener PCI).
  - [vxVME.c] mvmestd implementation for VxWorks Operating System. Does require cross compiler for the VxWorks target hardware processor and proper WindRiver license.

- VME modules
  - [lrs1190.c] LeCroy Dual-port memory ECL 32bits.
  - [lrs1151.c] LeCroy 16 ECL 32bits scalers.
  - [lrs2365.c] LeCroy Logic matrix.
  - [lrs2373.c] LeCroy Memory Lookup unit.
  - [sis3700.c] SIS FERA Fifo 32 bits.
  - [sis3801.c] SIS MultiChannel Scalers 32 channels.
  - [sis3803.c] SIS Standard 32 Scalers 32 bits.
  - [ps7106.c] Phillips Scientific Discriminator.
  - [ces8210.c] CES CAMAC crate controller.
  - [vmeio.c] Triumf VMEIO General purpose I/O 24bits.

6.3.3 USB drivers

This section is under development for the Wiener USB/CAMAC CCUSB controller. Support for Linux and XP is undergo. Please contact midas for further information.

For GPIB Linux support please refer to The Linux Lab Project

6.3.4 GPIB drivers

There is no specific GPIB driver part of the Midas package. But GPIB is used at Triumf under WindowsNT for several Slow Control frontends. The basic GPIB DLL library is provided by National Instrument. Please contact midas for further information.

For GPIB Linux support please refer to The Linux Lab Project

6.3.5 Other drivers

- **[Serial driver]** rs232.c communication routines.
- **[Network driver] tcpip.c/h** TCP/IP socket communication routines.
- **[SCSI driver]** Support for the jorway73a SCSI/CAMAC controller under Linux has been done by Greg Hackman (see CAMAC drivers).

Data format - Top - CAMAC and VME access function call

6.4 CAMAC and VME access function call

Supported hardware - Top - Midas build options and operation considerations

Midas defines its own set of CAMAC, VME and FASTBUS calls in order to unify the different hardware modules that it supports. This interface method permits to be totally hardware as well as OS **independent**. The same user code developed on a system can be used as a template for another application on a different operating system.

While the file mcstd.h (Midas Camac Standard) provides the interface for the CAMAC access, the file mvmestd.h (Midas VME Standard) is for the VME access. An extra CAMAC interface built on the top of **mcstd** provides the ESONE standard CAMAC calls (esone.c).

Refers to the corresponding directories under **/drivers** to find out what module of each family is already supported by the current Midas distribution. /drivers/divers contains older drivers which has not yet been converted to the latest API.

6.4.1 Midas CAMAC standard functions

Please refer to mcstd.h for function description.

6.4.2 ESONE CAMAC standard functions

Not all the functionality of ESONE standard have been fully tested

Please refer to esone.c for function description.

6.4.3 Midas VME standard functions

This API provides basic VME access through a **simple** set of functions. Refer to mvmestd.h for more specific information. mvme_open() contains a general access code sample summarizing most of the mvme commands.

6.4.4 Computer Busy Logic

A "computer busy logic" has to be implemented for a front-end to work properly. The reason for this is that some ADC modules can be re-triggered. If they receive more than one gate pulse before being read out, they accumulate the input charge that leads to wrong results. Therefore only one gate pulse should be sent to the ADC's, additional pulses must be blocked before the event is read out by the front-end. This operation is usually performed by a latch module, which is set by the trigger signal and reset by the computer after it has read out the event:

The output of this latch is shaped (limited in its pulse with to match the ADC gate width) and distributed to the ADC's. This scheme has two problems. The computer generates the reset signal, usually by two CAMAC output functions to a CAMAC IO unit. Therefore the duration of the pulse is a couple of ms. There is a non-negligible probability that during the reset pulse there is another hardware trigger. If this happens and both inputs of the latch are active, its function is undefined. Usually it generates several output pulses that lead to wrong ADC values. The second problem lies in the fact that the latch can be just reset when a trigger input is active. This can happen since trigger signals usually have a width of a few tens of nanoseconds. In this case the latch output signal does not carry the timing of the trigger signal, but the timing of the reset signal. The wrong timing of the output can lead to false ADC and TDC signals. To overcome this problem, a more elaborate scheme is necessary. One possible solution is the use of a latch module with edge-sensitive input and veto input. At PSI, the module "D. TRIGGER / DT102" can be used. The veto input is also connected to the computer:

Latched trigger layout.

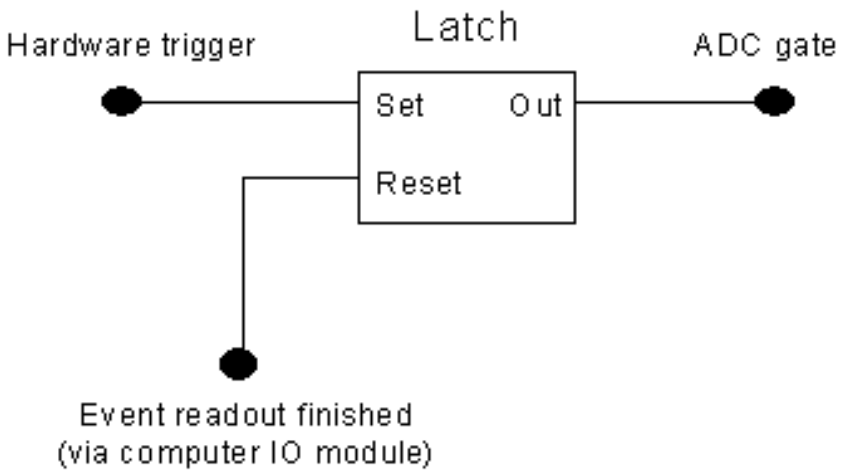


Figure 7: Latched trigger layout.

To reset this latch, following bit sequence is applied to the computer output (signals are displayed active low):

Improved Latched trigger layout.

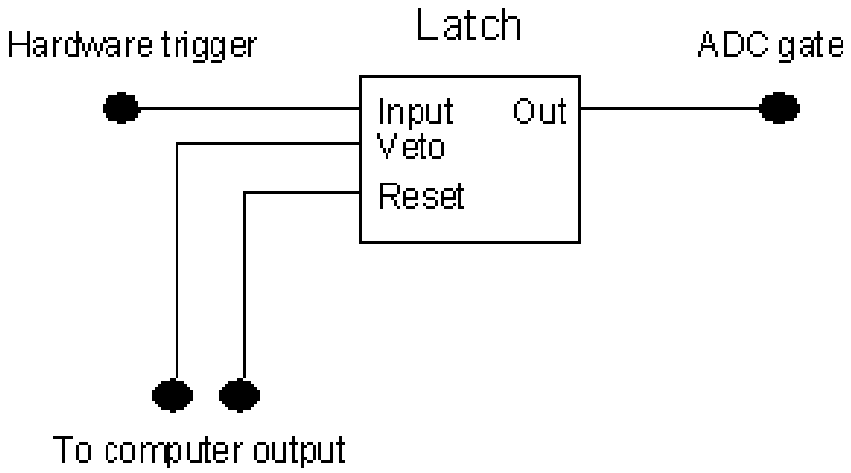


Figure 8: Improved Latched trigger layout.

The active veto signal during the reset pulse avoids that the latch can receive a "set" and a "reset" simultaneously. The edge sensitive input ensures that the latch can only trigger on a leading edge of a trigger signal, not on the removing of the veto signal. This ensures that the timing of the trigger is always carried at the ADC/TDC gate signal.

Veto Timing.

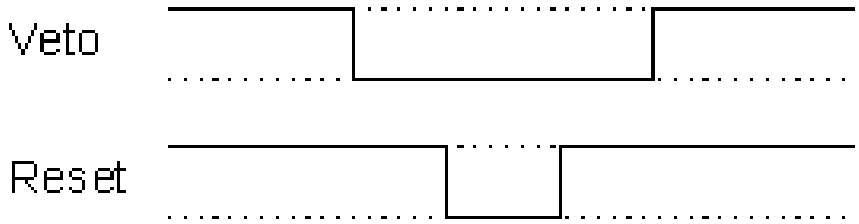


Figure 9: Veto Timing.

Supported hardware - Top - Midas build options and operation considerations

6.5 Midas build options and operation considerations

CAMAC and VME access function call - Top - Midas Code and Libraries

The section covers the Building Options for customization of the DAQ system as well as the different Environment variables options for its operation.

6.5.1 Building Options

- By default Midas is build with a minimum of pre-compiler flags. But the Makefile contains options for the user to apply customization by enabling internal options already available in the package.
  - YBOS_VERSION_3_3 , EVID_TWIST , INCLUDE_FTPLIB , INCLUDE_ZLIB , SPECIFIC_OS_PRG
- Other flags are avaiable at the application level:
  - HAVE_CAMAC , HAVE_ROOT , HAVE_HBOOK , HAVE_MYSQL , USE_EVENT_CHANNEL , DM_DUAL_THREAD , USE_INT
- By default the midas applications are built for use with dynamic library **libmidas.so**. If static build is required the whole package can be built using the option **static**.

```
> make static
```

- The basic Midas package builds without external package library reference. But it does try to build an extra core analyzer application to be used in conjunction with ROOT if $ROOTSYS is found. This is required ONLY if the examples/experiment makefile is used for generating a complete Midas/ROOT analyzer application.

- In case of HBOOK/PAW analyzer application, the build should be done from examples/hbookexpt directory and the environment variable CERNLIB_PACK should be pointing to a valid cernpacklib.a library.

- For development it could be useful to built individual application in static. This can be done using the USERFLAGS option such as:

```
> rm linux/bin/mstat; make USERFLAGS=-static linux/bin/mstat
```

- The current OS support is done through fix flag established in the general Makefile . Currently the OS supported are:
  - **OS_OSF1** , **OS_ULTRIX** , **OS_FREEBSD** , **OS_LINUX** , **OS_SOLARIS**.

- For **OS_IRIX** please contact Pierre. The **OS_VMS** is not included in the Makefile as it requires a particular makefile and since several years now the VMS support has been dropped.

```
OSFLAGS = -DOS_LINUX ...
```

- **OSFLAGS [2.0.0]** For 32 bit built, the OSFLAGS should contains the -m32. By default this flag is not enabled. It has to be applied to the Makefile for the frontend examples too.

```
# add to compile midas in 32-bit mode
# OSFLAGS += -m32
```

- Other OS supported are:
  - OS_WINNT : See file makefile.nt.
  - OS_VXWORKS : See file makefile.ppc_tri.

6.5.2 USERFLAGS

This flag can be used at the command prompt for individual application built.

```
make USERFLAGS=-static linux/bin/mstat
```

6.5.3 MIDAS_PREF_FLAGS

This flag is for internal global Makefile preference. Included in the **OSFLAGS**.

```
MIDAS_PREF_FLAGS  = -DYBOS_VERSION_3_3 -DEVID_TWIST
```

6.5.4 HAVE_CAMAC

This flag enable the CAMAC RPC service within the frontend code. The application mcnaf task and the web CNAF page are by default not CAMAC enabled (HAVE_-CAMAC undefined).

6.5.5 HAVE_ROOT

This flag is used for the midas analyzer task in the case **ROOT** environment is required. An example of the makefile resides in **examples/experiment/Makefile**. This flag is enabled by the presence of a valid **ROOTSYS** environment variable. In the case **ROOTSYS** is not found the analyzer is build without **ROOT** support. In this later case, the application rmidas task will be missing. refer to MIDAS Analyzer for further details.

6.5.6 HAVE_HBOOK

This flag is used for **examples/hbookexpt/Makefile** for building the midas analyzer task against **HBOOK** and **PAW**. The path to the cernlib is requested and expected to be found under /cern/pro/lib (see makefile). This can always be overwritten during the makefile using the following command:

```
make CERNLIB_PACK=<your path>/libpacklib.a
```

6.5.7 HAVE_MYSQL

This flag is used for the mlogger task to building the application with *mySQL* support. The build requires to have access to the mysql include files as well as the mysql library.

- For operation of the analyzer **without HBOOK** or **ROOT**, refer to MIDAS Analyzer for further details.

6.5.8 SPECIFIC_OS_PRG

This flag is for internal Makefile preference. Used in particular for additional applications build based on the OS selection. In the example below mspeaker, mlxspeaker tasks and dio task are built only under OS_LINUX.

```
SPECIFIC_OS_PRG = $(BIN_DIR)/mlxspeaker_task $(BIN_DIR)/dio_task
```

6.5.9 INCLUDE_FTPLIB

FTP support "INCLUDE_FTPLIB" Application such as the mlogger task, lazylogger task can use the ftp channel for data transfer.

6.5.10 INCLUDE_ZLIB

The applications lazylogger task, mdump task can be built with **zlib.a** in order to gain direct access to the data within a file with extension **mid.gz** or **ybs.gz**. By default this option is disabled except for the system analyzer core code **mana.c**.

```
make USERFLAGS=-DINCLUDE_ZLIB linux/lib/ybos.o
make USERFLAGS=-static linux/bin/mdump
```

6.5.11 YBOS_VERSION_3_3

The default built for ybos support is version 4.0. If lower version is required include **YBOS_VERSION_3_3** during compilation of the ybos.c

```
make USERFLAGS=-DYBOS_VERSION_3_3 linux/lib/ybos.o
```

6.5.12 DM_DUAL_THREAD

Valid only under VxWorks. This flag enable the dual thread task when running the frontend code under VxWorks. The main function calls are the dm_xxxx in midas.c (Contact Pierre for more information).

6.5.13 USE_EVENT_CHANNEL

To be used in conjunction with the DM_DUAL_THREAD.

6.5.14 USE_INT

In mfe.c. Enable the use of interrupt mechanism. This option is so far only valid under VxWorks Operating system. (Contact Stefan or Pierre for further information).

6.5.15 Environment variables

Midas uses a several environment variables to facilitate the different application startup.

6.5.15.1 MIDASSYS From version 1.9.4 this environmental variable is required. It should point to the main path of the installed Midas package. The application odbedit will generate a warning message in the case this variable is not defined.

6.5.15.2 MIDAS_EXPTAB This variable specify the location of the **exptab** file containing the predefined midas experiment. The default location is for OS_UNIX: /etc, /. For OS_WINNT: \system32, \system.

6.5.15.3 MIDAS_SERVER_HOST This variable predefines the names of the host on which the Midas experiment shared memories are residing. It is needed when connection to a remote experiment is requested. This variable is valid for Unix as well as Windows OS.

6.5.15.4 MIDAS_EXPT_NAME This variable predefines the name of the experiment to connect by default. It prevents the requested application to ask for the experiment name when multiple experiments are available on the host or to add the -e <expt_name> argument to the application command. This variable is valid for Unix as well as Windows OS.

6.5.15.5 MIDAS_DIR This variable predefines the LOCAL directory path where the shared memories for the experiment are located. It supersede the host_name and the expt_name as well as the MIDAS_SERVER_HOST and MIDAS_EXPT_NAME as a given directory path can only refer to a single experiment.

6.5.15.6 MCHART_DIR This variable is ... for later... This variable is valid only under Linux as the -D is not supported under WindowsXX

CAMAC and VME access function call - Top - Midas Code and Libraries

6.6 Midas Code and Libraries

Midas build options and operation considerations - Top - Frequently Asked Questions

This section covers several aspect of the Midas system.

6.6.1 State Codes & Transition Codes

- State Codes : These number will be apparent in the ODB under the ODB /RunInfo Tree.

  - STATE_STOPPED
  - STATE_PAUSED
  - STATE_RUNNING

- Transition Codes These number will be apparent in the ODB under the ODB /RunInfo Tree.
  - TR_START
  - TR_STOP
  - TR_PAUSE
  - TR_RESUME

6.6.2 Midas Data Types

Midas defined its own data type for OS compatibility. It is suggested to use them in order to insure a proper compilation when moving code from one OS to another. *float* and *double* retain OS definition.

- BYTE unsigned char
- WORD unsigned short int (16bits word)
- DWORD unsigned 32bits word
- INT signed 32bits word
- BOOL OS dependent.

When defining a data type either in the frontend code for bank definition or in user code to define ODB variables, Midas requires the use of its own data type declaration. The list below shows the main Type IDentification to be used (refers to Midas Define for complete listing):

- TID_BYTE unsigned byte 0 255
- TID_SBYTE signed BYTE -128 127
- TID_CHAR single character 0 255
- TID_WORD two BYTE 0 65535
- TID_SHORT signed WORD -32768 32767
- TID_DWORD four bytes 0 2**32-1

- TID_INT signed DWORD -2**31 2**31-1
- TID_BOOL four bytes bool 0 1
- TID_FLOAT four bytes float format
- TID_DOUBLE eight bytes float format

6.6.3 Midas bank examples

There are several examples under the Midas source code that you can check. Please have a look at

- Frontend code midas/examples/experiment/frontend.c etc...
- Backend code midas/examples/experiment/analyzer.c etc...

6.6.4 YBOS Bank Types

YBOS defines several type but all types should be 4 bytes aligned. Distinction of signed and unsigned is not done. When mixing MIDAS and YBOS in the frontend for RO_ODB see The Equipment structure make sure the bank types are compatible (see also YBOS Define)

- **I1_BKTYPE** Bank of Bytes
- **I2_BKTYPE** Bank of 2 bytes data
- **I4_BKTYPE** Bank of 4 bytes data
- **F4_BKTYPE** Bank of float data
- **D8_BKTYPE** Bank of double data
- **A1_BKTYPE** Bank of ASCII char

6.6.5 YBOS bank examples

Basic examples using YBOS banks are available in the midas tree under examples/ybosexpt.

- **Frontend code** Example 1, 2 shows the bank creation with some CAMAC acquisition.

```
-------- example 1 -------- Simple 16 bits bank construction

void read_cft (DWORD *pevent)
{
  DWORD *pbkdat, slot;

  ybk_create((DWORD *)pevent, "TDCP", I2_BKTYPE, &pbkdat);
  for (slot=FIRST_CFT;slot<=LAST_CFT;slot++)
    {
      cami(3,slot,1,6,(WORD *)pbkdat);
      ((WORD *)pbkdat)++;
      cam16i_rq(3,slot,0,4,(WORD **)&pbkdat,16);
    }
  ybk_close((DWORD *)pevent, I2_BKTYPE, pbkdat);
  return;
}
-------- example 2 -------- Simple 32bit bank construction
{
  DWORD *pbkdat;

  ybk_create((DWORD *)pevent, "TICS", I4_BKTYPE, &pbkdat);
  camo(2,22,0,17,ZERO);
  cam24i_r(2,22,0,0,(DWORD **) &pbkdat,10);
  cam24i_r(2,22,0,0,(DWORD **) &pbkdat,10);
  cam24i_r(2,22,0,0,(DWORD **) &pbkdat,10);
  cam24i_r(2,22,0,0,(DWORD **) &pbkdat,10);
  cam24i_r(2,22,0,0,(DWORD **) &pbkdat,9);
  ybk_close((DWORD *)pevent, I4_BKTYPE, pbkdat);
  return 0;
}
```

Example 3 shows a creation of an EVID bank containg a duplicate of the midas header. As the Midas header is stripped out of the event when data are logger, it is necessary to compose such event to retain event information for off-line analysis. Uses of predefined macros (see Midas Code and Libraries) are available in order to extract from a precomposed Midas event the internal header fields i.e. Event ID, Trigger mask, Serial number, Time stamp. In this EVID bank we added the current run number which is retrieve by the frontend at the begin of a run.

```
-------- example 3 -------- Full equipment readout function

INT read_cum_scaler_event(char *pevent, INT off)
{
  INT i;
```

```
  DWORD *pbkdat, *pbktop, *podbvar;

  ybk_init((DWORD *) pevent);

  // collect user hardware SCALER data
  ybk_create((DWORD *)pevent, "EVID", I4_BKTYPE, (DWORD *)(&pbkdat));
  *(pbkdat)++ = gbl_tgt_counter++;                      // event counter
  *((WORD *)pbkdat) = EVENT_ID(pevent);     ((WORD *)pbkdat)++;
  *((WORD *)pbkdat) = TRIGGER_MASK(pevent); ((WORD *)pbkdat)++;
  *(pbkdat)++ = SERIAL_NUMBER(pevent);
  *(pbkdat)++ = TIME_STAMP(pevent);
  *(pbkdat)++ = gbl_run_number;                         // run number
  ybk_close((DWORD *)pevent, pbkdat);

  // BEGIN OF CUMULATIVE SCALER EVENT
  ybk_create((DWORD *)pevent, "CUSC", I4_BKTYPE, (DWORD *)(&pbkdat));
  for (i=0 ; i<NSCALERS ; i++){
    *pbkdat++ = scaler[i].cuval[0];
    *pbkdat++ = scaler[i].cuval[1];
  }

  ybk_close(DWORD *)pevent, I4_BKTYPE, pbkdat);
  // END OF CUMULATIVE SCALER EVENT

  // event in bytes for Midas
  return (ybk_size ((DWORD *)pevent));
}
```

- **Backend code** If the data logging is done through YBOS format (see ODB /Logger Tree Format) the events on the storage media will have been stripped from the MIDAS header used for transfering the event from the frontend to the backend. This means the logger data format is a "TRUE" YBOS format. Uses of standard YBOS library is then possible.

```
--- Example of YBOS bank extraction ----

void process_event(HNDLE hBuf, HNDLE request_id, EVENT_HEADER *pheader, void *pevent)
{
    INT status;
    DWORD *plrl, *pybk, *pdata, bklen, bktyp;
    char  banklist[YB_STRING_BANKLIST_MAX];

    // pointer to data section
    plrl = (DWORD *)       pevent;

    // Swap event
    yb_any_event_swap(FORMAT_YBOS,plrl);

    // bank name given through argument list
    if ((status = ybk_find (plrl, sbank_name, &bklen, &bktyp, (void *)&pybk)) == YB_SUCCESS)
    {
      // given bank found in list
      status = ybk_list (plrl, banklist);
      printf("#banks:%i Bank list:-%s-\n",status,banklist);
      printf("Bank:%s - Length (I*4):%i - Type:%i - pBk:0x%p\n",sbank_name, bklen, bktyp, pybk);
```

```
      // check id EVID found in event for id and msk selection
      if ((status = ybk_find (plrl, "EVID", &bklen, &bktyp, (void *)&pybk)) == YB_SUCCESS)
      {
        pdata = (DWORD *)((YBOS_BANK_HEADER *)pybk + 1);
  ...
      }

     // iterate through the event
     pybk = NULL;
     while ((bklen = ybk_iterate(plrl, &pybk, (void *)&pdata))
                  && (pybk != NULL))
       printf("bank length in 4 bytes unit: %d\n",bklen);

    }
    else
    {
      status = ybk_list (plrl, banklist);
      printf("Bank -%s- not found (%i) in ",sbank_name, status);
      printf("#banks:%i Bank list:-%s-\n",status,banklist);
    }
    ...
    ... ...
}
```

6.6.6 Midas Code and Libraries

The Midas libraries are composed of 5 main source code and their corresponding header files.

1. The midas.h & midas.c : Midas abstract layer.
2. The msystem.h & system.c : Midas function implementation.
3. The mrpc.h & mrpc.c : Midas RPC functions.
4. The odb.c : Online Database functions.
5. The ybos.h & ybos.c : YBOS specific functions.

Within these files, all the functions have been categorized depending on their scope.

- **al_xxx**(...) : Alarm system calls
- **bk_xxx**(...) : Midas bank manipulation calls
- **bm_xxx**(...) : Buffer management calls
- **cm_xxx**(...) : Common system calls

- **db_xxx**(...) : Database managment calls
- **el_xxx**(...) : Electronic Log book calls
- **hs_xxx**(...) : History manipulation calls
- **ss_xxx**(...) : System calls
- **ybk_xxx**(...) : YBOS bank manipulation

6.6.7 MIDAS Macros

Several group of MACROs are available for simplifying user job on setting or getting Midas information. They are also listed in the Midas Code and Libraries. All of them are defined in the Midas Macros, System Macros, YBOS Macros header files.

- **Message Macros**. These Macros compact the 3 first arguments of the cm_msg() call. It replaces the type of message, the routine name and the line number in the C-code. See example in cm_msg().
  - MERROR : For error (MT_ERROR, __FILE__, __LINE__)
  - MINFO : For info (MT_INFO, __FILE__, __LINE__)
  - MDEBUG : For debug (MT_DEBUG, __FILE__, __LINE__)
  - MUSER : Produced by interactive user (MT_USER, __FILE__, __LINE_-_)
  - MLOG : Info message which is only logged (MT_LOG, __FILE__, __-LINE__)
  - MTALK : Info message for speech system (MT_TALK, __FILE__, __-LINE__)
  - MCALL : Info message for telephone call (MT_CALL, __FILE__, __-LINE__)
- **DAQ Event/LAM Macros**. To be used in the frontend/analyzer code.
  - **CAMAC LAM manipulation**. These Macros are used in the frontend code to interact with the LAM register. Usualy the CAMAC Crate Controler has the feature to register one bit per slot and be able to present this register to the user. It may even have the option to mask off this register to allow to set a "general" LAM register containing either "1" (At least one LAM from the masked LAM is set) or "0" (no LAM set from the maksed LAM register). The poll_event() uses this feature and return a variable which contains a bit-wise value of the current LAM register in the Crate Controller.

  - LAM_SOURCE
  - LAM_STATION
  - LAM_SOURCE_CRATE
  - LAM_SOURCE_STATION

- **BYTE swap manipulation**. These Macros can be used in the backend analyzer when **little-endian/big-endian** are mixed in the event.
  - WORD_SWAP
  - DWORD_SWAP
  - QWORD_SWAP

- **MIDAS Event Header manipulation**. Every event travelling through the Midas system has a "Event Header" containing the minimum information required to identify its content. The size of the header has been kept as small as possible in order to minimize its impact on the data rate as well as on the data storage requirment. The following macros permit to read or override the content of the event header as long as the argument of the macro refers to the top of the Midas event (pevent). This argument is available in the frontend code in any of the user readout function (pevent). It is also available in the user analyzer code which retrieve the event and provide directly access to the event header (pheader) and to the user part of the event (pevent). Sub-function using pevent would then be able to get back the the header through the use of the macros.
  - TRIGGER_MASK
  - EVENT_ID
  - SERIAL_NUMBER
  - TIME_STAMP
    * from examples/experiment/adccalib.c

```
INT adc_calib(EVENT_HEADER *pheader, void *pevent)
{
  INT    i, n_adc;
  WORD   *pdata;
  float  *cadc;

  // look for ADC0 bank, return if not present
  n_adc = bk_locate(pevent, "ADC0", &pdata);
  if (n_adc == 0 || n_adc > N_ADC)
    return 1;

  // create calibrated ADC bank
  bk_create(pevent, "CADC", TID_FLOAT, &cadc);
  ...
}
```

    * from examples/experiment/frontend.c

```
INT read_trigger_event(char *pevent, INT off)
{
  WORD *pdata, a;
  INT  q, timeout;

  // init bank structure
  bk_init(pevent);
  ...
}
```

– Frontend C-code fragment from running experiment:

```
INT read_ge_event(char *pevent, INT offset)
{
  static WORD *pdata;
  INT i, x, q;
  WORD temp;

  // Change the time stamp in millisecond for the Super event
  TIME_STAMP(pevent) = ss_millitime();

  bk_init(pevent);
  bk_create(pevent, "GERM", TID_WORD, &pdata);
  ...
}
```

– Frontend C-code fragment from running experiment

```
...
lam = *((DWORD *)pevent);

if (lam & LAM_STATION(JW_N))
{
  ...
  // compose event header
  TRIGGER_MASK(pevent) = JW_MASK;
  EVENT_ID(pevent)     = JW_ID;
  SERIAL_NUMBER(pevent)= eq->serial_number++;
  // read MCS event
  size = read_mcs_event(pevent);
  // Correct serial in case event is empty
  if (size == 0)
    SERIAL_NUMBER(pevent) = eq->serial_number--;
  ...
}
...
```

6.6.7.1 YBOS library Exportable ybos functions through inclusion of ybos.h

Midas build options and operation considerations - Top - Frequently Asked Questions

6.7 Frequently Asked Questions

Midas Code and Libraries - Top - Data format

Feel free to ask questions to one of us (Stefan Ritt , Pierre-Andre Amaudruz) or visit the Midas Forum

1. **Why the CAMAC frontend generate a core dump (linux)?**
   - If you're not using a Linux driver for the CAMAC access, you need to start the CAMAC frontend application through the task launcher first. See dio task or mcnaf task. This task laucher will grant you access permission to the IO port mapped to your CAMAC interface.
2. **Where does Midas log file resides?**
   - As soon as any midas application is started, a file midas.log is produce. The location of this file depends on the setup of the experiment.

   (a) if **exptab** is present and contains the experiment name with the corresponding directory, this is where the file **midas.log** will reside.

   (b) if the midas logger mlogger task is running the midas.log will be in the directory pointed by the "Data Dir" key under the /logger key in the ODB tree.

   (c) Otherwise the file midas.log will be created in the current directory in which the Midas application is started.
3. **How do I protected my experiment from being controlled by aliases?**
   - Every experiment may have a dedicated password for accessing the experiment from the web browser. This is setup through the ODBedit program with the command **webpass**. This will create a **Security** tree under **/Experiment** with a new key **Web Password** with the encrypted word. By default Midas allows Full Read Access to all the Midas Web pages. Only when modification of a Midas field the web password will be requested. The password is stared as a cookie in the target web client for 24 hours See ODB /Experiment Tree.
   - Other options of protection are described in ODB /Experiment Tree which gives to dedicated hosts access to ODB or dedicated programs.
4. **Can I compose my own experimental web page?**
   - Only under 1.8.3 though. You can create your own html code using your favorite HMTL editor. By including custom Midas Tags, you will have access to any field in the ODB of your experiment as well as the standard button for start/stop and page switch. See mhttpd task , Custom page.
5. **How do I prevent user to modify ODB values while the run is in progress?**

- By creating the particular **/Experiment/Lock** when running/ ODB tree, you can include symbolic links to any odb field which needs to be set to **Read Only** field while the run state is on. See ODB /Experiment Tree.

6. **Is there a way to invoke my own scripts from the web?**

   - Yes, by creating the ODB tree **/Script** every entry in that tree will be available on the Web status page with the name of the key. Each key entry is then composed with a list of ODB field (or links) starting with the executable command followed by as many arguments as you wish to be passed to the script. See ODB /Script Tree.

7. **I've seen the ODB prompt displaying the run state, how do you do that?**

   - Modify the **/System/prompt** field. The "S" is the trick.

```
Fri> odb -e bnmr1 -h isdaq01
[host:expt:Stopped]/cd /System/
[host:expt:Stopped]/System>ls
Clients
Client Notify                   0
Prompt                          [%h:%e:%S]%p
Tmp
[host:expt:Stopped]/System
[host:expt:Stopped]/Systemset prompt [%h:%e:%S]%p>
[host:expt:Stopped]/System>ls
Clients
Client Notify                   0
Prompt                          [%h:%e:%S]%p>
Tmp
[host:expt:Stopped]/System>set Prompt [%h:%e:%s]%p>
[host:expt:S]/System>set Prompt [%h:%e:%S]%p>
[host:expt:Stopped]/System>
```

8. **I've setup the alarm on one parameter in ODB but I can't make it trigger?**

   - The alarm scheme works only under **ONLINE**. See ODB /RunInfo Tree for **Online Mode**. This flag may have been turned off due to analysis replay using this ODB. Set this key back to 1 to get the alarm to work again.

9. **How do I extend an array in ODB?**

   - When listing the array from ODB with the -l switch, you get a column indicating the index of the listed array. You can extend the array by setting the array value at the new index. The intermediate indices will be fill with the default value depending on the type of the array. This can easly corrected by using the wildcard to access all or a range of indices.

```
[local:midas:S]/>mkdir tmp
[local:midas:S]/>cd tmp
[local:midas:S]/tmp>create int number
[local:midas:S]/tmp>create string foo
String length [32]:
[local:midas:S]/tmp>ls -l
```

```
Key name                          Type    #Val  Size  Last Opn Mode Value
---------------------------------------------------------------------------
number                            INT     1     4     >99d 0   RWD  0
foo                               STRING  1     32    1s   0   RWD
[local:midas:S]/tmp>set number[4] 5
[local:midas:S]/tmp>set foo[3]
[local:midas:S]/tmp>ls -l
Key name                          Type    #Val  Size  Last Opn Mode Value
---------------------------------------------------------------------------
number                            INT     5     4     12s  0   RWD
                                          [0]             0
                                          [1]             0
                                          [2]             0
                                          [3]             0
                                          [4]             5
foo                               STRING  4     32    2s   0   RWD
                                          [0]
                                          [1]
                                          [2]
                                          [3]
[local:midas:S]/tmp>set number[1..3] 9
[local:midas:S]/tmp>set foo[2] "A default string"
[local:midas:S]/tmp>ls -l
Key name                          Type    #Val  Size  Last Opn Mode Value
---------------------------------------------------------------------------
number                            INT     5     4     26s  0   RWD
                                          [0]             0
                                          [1]             9
                                          [2]             9
                                          [3]             9
                                          [4]             5
foo                               STRING  4     32    3s   0   RWD
                                          [0]
                                          [1]
                                          [2]             A default string
                                          [3]
```

1. How do I ...

   - ...
     Midas Code and Libraries - Top - Data format

6.8 Components

Introduction - Top - Quick Start

Midas system is based on a modular scheme that allows scalability and flexibility. Each component's operation is handled by a sub-set of functions. but all the components are grouped in a single library (libmidas.a, libmidas.so(UNIX), midas.dll(NT)).

The overall C-code is about 80'000 lines long and makes up over 450 functions (version 1.9.0). But from a user point of view, only a subset of these routines are needed for most operations.

Each Midas component is briefly described below but throughout the documentation more detailed information will be given regarding each of their capabilities. All these components are available from the "off-the-shelf" package. Basic components such as the Buffer Manager, Online Database, Message System, Run Control are by default operationals. The other needs are to be enabled by the user simply by either starting an application or by activation of the component through the Online Database. A general picture of the Midas system is displayed below.

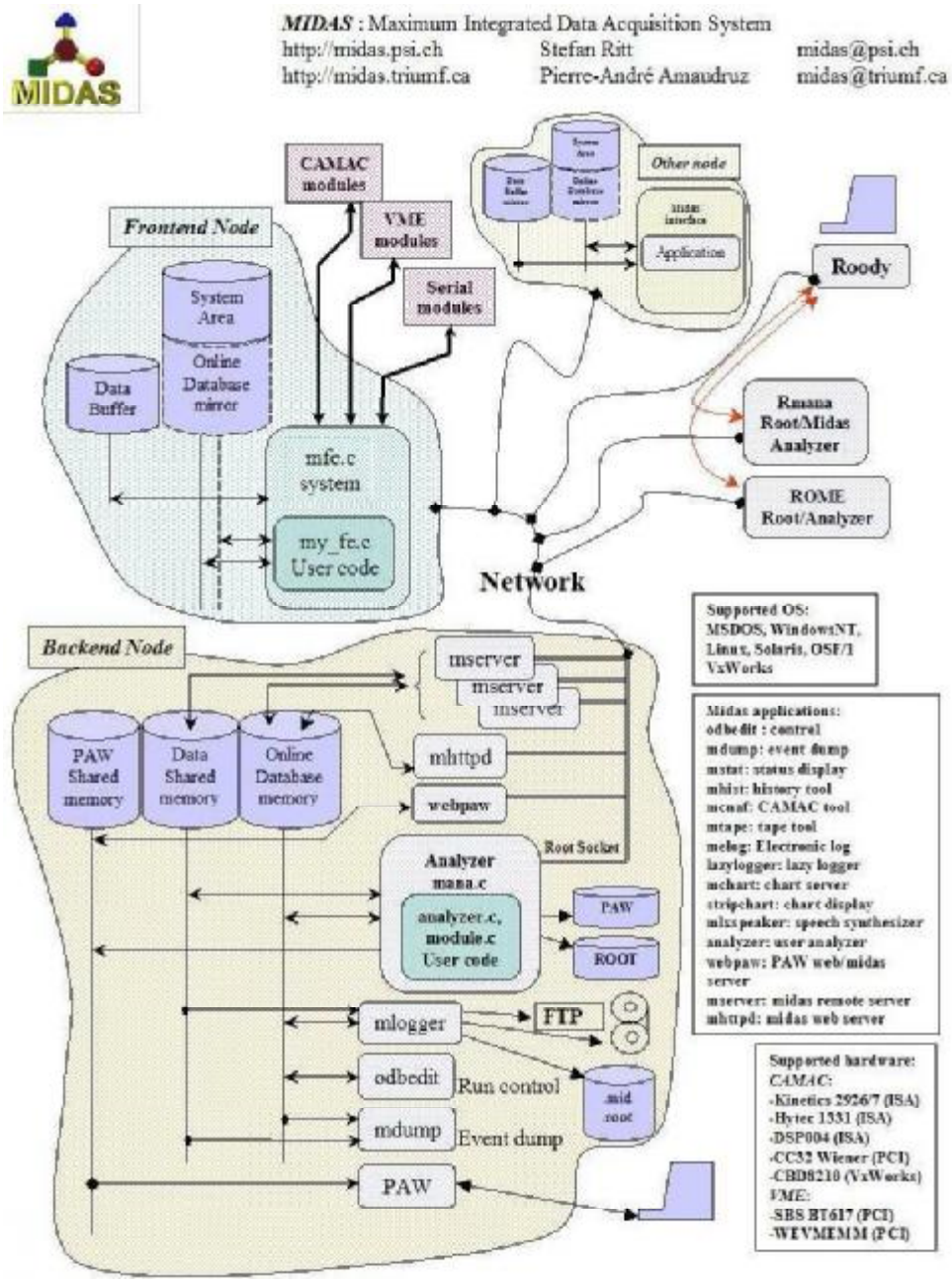


Figure 10: Components

The main elements of the **Midas** package are listed below with a short description of its functionality.

- **Buffer Manager** Data flow and messages passing mechanism.
- **Message System** Specific Midas messages flow.
- **Online Database** Central information area.
- **Frontend** Acquisition code.
- **Midas Server** Remote access server (RPC server).
- **Data Logger** Data storage.
- **Analyzer** Data analyzer.
- **Run Control** Data flow control.
- **Slow Control** system Device monitoring and control.
- **History system** Event history storage and retrival.
- **Alarm System** Overall system and user alarm.
- **Electronic Logbook** Online User Logbook.

6.8.1 Buffer Manager

The "buffer manager" consists of a set of library functions for event collection and distribution. A buffer is a shared memory region in RAM, which can be accessed by several processes, called "clients". Processes sending events to a buffer are called "producers", processes reading events are called "consumers".

A buffer is organized as a FIFO (First-In-First-Out) memory. Consumers can specify which type of events they want to receive from a buffer. For this purpose each event contains a MIDAS header with an event ID and other pertinent information.

Buffers can be accessed locally or remotely via the MIDAS server. The data throughput for a local configuration composed of one producer and two consumers is about 10MB/sec on a 200 MHz Pentium PC running Windows NT. In the case of remote access, the network may be the essential speed limitation element.

A common problem in DAQ systems is the possible crash of a client, like a user analyzer. This can cause the whole system to hang up and may require a restart of the DAQ inducing a lost of time and eventually precious data. In order to address this problem, a special watchdog scheme has been implemented. Each client attached to the buffer

manager signals its presence periodically by storing a time stamp in the shared memory. Every other client connected to the same buffer manager can then check if the other parties are still alive. If not, proper action is taken consisting of removing the dead client hooks from the system leaving the system in a working condition.

6.8.2 Message System

Any client can produce status or error messages with a single call using the MIDAS library. These messages are then forwarded to any other clients who maybe susceptible to receive these messages as well as to a central log file system. The message system is based on the buffer manager scheme. A dedicated buffer is used to receive and distribute messages. Predefined message type contained in the Midas library covers most of the message requirement.

6.8.3 Online Database

In a distributed DAQ environment configuration data is usually stored in several files on different computers. MIDAS uses a different approach. All relevant data for a particular experiment are stored in a central database called "Online Database" (ODB). This database contains run parameters, logging channel information, condition parameters for front-ends and analyzers and slow control values as well as status and performance data.

The main advantage of this concept is that all programs participating in an experiment have full access to these data without having to contact different computers. The possible disadvantage could be the extra load put on the particular host serving the ODB.

The ODB is located completely in shared memory of the back-end computer. The access function to an element of the ODB has been optimized for speed. Measurement shows that up to 50,000 accesses per second local connection and around 500 accesses per second remotely over the MIDAS server can be obtained.

The ODB is hierarchically structured, similar to a file system, with directories and sub-directories. The data is stored in pairs of a key/data, similar to the Windows NT registry. Keys can be dynamically created and deleted. The data associated with a key can be of several types such as: byte, words, double words, float, strings, etc. or arrays of any of those. A key can also be a directory or a symbolic link (like on Unix).

The Midas library provides a complete set of functions to manage and operate on these keys. Furthermore any ODB client can register a Hot Link between a local C-structure and a element of the ODB. Whenever a client (program) changes a value in this subtree, the C-structure automatically receives an update of the changed data. Additionally, a client can register a callback function which will be executed as soon as the hot-link's update has been received. For more information see ODB Structure.

6.8.4 Midas Server

For remote access to a MIDAS experiment a remote procedure call (RPC) server is available. It uses an optimized MIDAS RPC scheme for improved access speed. The server can be started manually or via inetd (UNIX) or as a service under Windows NT. For each incoming connection it creates a new sub-process which serves this connection over a TCP link. The Midas server not only serves client connection to a given experiment, but takes the experiment's name as a parameter meaning that only one Midas server is necessary to manage several experiments on the same node.

6.8.5 Frontend

The *frontend* program refers to a task running on a particular computer which has access to hardware equipment. Several *frontends* can be attached simultaneously to a given experiment. Each *frontend* can be composed of multiple *Equipment*. *Equipment* is a single or a collection of sub-task(s) meant to collect and regroup logically or physically data under a single and uniquely identified event.

This program is composed of a general framework which is experiment independent, and a set of template routines for the user to fill. This program will:

- Register the given *Equipment(s)* list to the Midas system.
- Provide the mean of collecting data from hardware source defined in each equipment.
- Gather these data in a known format (Fixed, Midas, Ybos) for each equipment.
- Sendsthese data to the buffer manager.
- Collect periodically statistic of the acquisition task and send it to the Online Database.

The frontend framework takes care of sending events to the buffer manager and optionally a copy to the ODB. A "Data cache " in the frontend and on the server side reduces the amount of network operations pushing the transfer speed closer to the physical limit of the network configuration.

The data collection in the frontend framework can be triggered by several mechanisms. Currently the frontend supports four different kind of event trigger:

- *Periodic events*: Scheduled event based on a fixed time interval. They can be used to read information such as scaler values, temperatures etc.
- *Polled events*: Hardware trigger information read continuously which in turns if the signal is asserted it will trigger the equipment readout.
  - *LAM events*: Generated only when pre-defined LAM is asserted:

- *Interrupt events*: Generated by particular hardware device supporting interrupt mode.
- *Slow Control events*: Special class of events that are used in the slow control system.

Each of these types of triggering can be enabled/activated for a particular experiment state, Transition State or a combination of any of them. Examples such as "read scaler event only when running" or "read periodic event if a state is not paused and on all transitions" are possible.

Dedicated header and library files for hardware access to CAMAC, VME, Fastbus, GPIB and RS232 are part of Midas distribution set. For more information see Frontend code.

6.8.6 Data Logger

The data logger is a client usually running on the backend computer (can be running remotely but performance may suffer) receiving events from the buffer manager and saving them onto disk, tape or via FTP to a remote computer. It supports several parallel logging channels with individual event selection criteria. Data can currently be written in five different formats: *MIDAS binary, YBOS binary, ASCII, ROOT and DUMP* (see Midas format, YBOS format).

Basic functionality of the logger includes:

- Run Control based on:
  - event limit
  - recorded byte limit
  - logging device full.
- Logging selection of particular events based on Event Identifier.
- Auto restart feature allowing logging of several runs of a given size without user intervention.
- Recording of ODB values to a so called History system
- Recording of the ODB to all or individual logging channel at the beginning and end of run state as well as to a separate disk file in a ASCII format. For more information see ODB /Logger Tree.

6.8.7 Analyzer

As in the front-end section, the analyzer provided by Midas is a framework on which the user can develop his/her own application. This framework can be built for private analysis (no external analyzer hooks) or specific analysis packages such as HBOOK, ROOT from the CERN (none of those libraries are included in the MIDAS distribution). The analyzer takes care of receiving events (a few lines of code are necessary to receive events from the buffer manager), initializes the HBOOK or ROOT system and automatically books N-tuples/TTree for all events. Interface to user routines for event analysis is provided.

The analyzer is structured into "stages", where each stage analyzes a subset of the event data. Low level stages can perform ADC and TDC calibration,while high level stages can calculate "physics" results. The same analyzer executable can be used to run online (receive events from the buffer manager) and off-line (read events from file). When running online, generated N-tuples/TTree are stored in a ring-buffer in shared memory. They can by analyzed with PAW without stopping the run. For ROOT please refer to the documentation ...

When running off-line, the analyzer can read MIDAS binary files, analyze the events, add calculated data for each event and produce a HBOOK RZ output file which can be read in by PAW later. The analyzer framework also supports analyzer parameters. It automatically maps C-structures used in the analyzer to ODB records via Hot Link. To control the analyzer, only the values in the ODB have to be changed which get automatically propagated to the analyzer parameters. If analysis software has been already developed, Midas provides the functionality necessary to interface the analyzer code to the Midas data channel. Support for languages such as C, FORTRAN, PASCAL is available.

6.8.8 Run Control

As mentioned earlier, the Online Database (ODB) contains all the pertinent information regarding an experiment. For that reason a run control program requires only to access the ODB. A basic program supplied in the package called ODBEdit provides a simple and safe mean for interacting with ODB. Through that program essentially all the flexibility of the ODB is available to the user's fingertips.

Three "Run State" define the state of Midas *Stopped*, *Paused*, *Running*. In order to change from one state to another, Midas provides four basic "Transition" function *Tr_Start*, *Tr_pause*, *Tr_resume*, *Tr_Stop*. During these transition periods, any Midas client register to receive notification of such message will be able to perform its task within the overall run control of the experiment.

In Order to provide more flexibility to the transition sequence of all the midas clients connected to a given experiment, each transition function has a *transition sequence number* attached to it. This transition sequence is used to establish within a given transition the order of the invocation of the Midas clients (from the lower seq.# to the

largest).

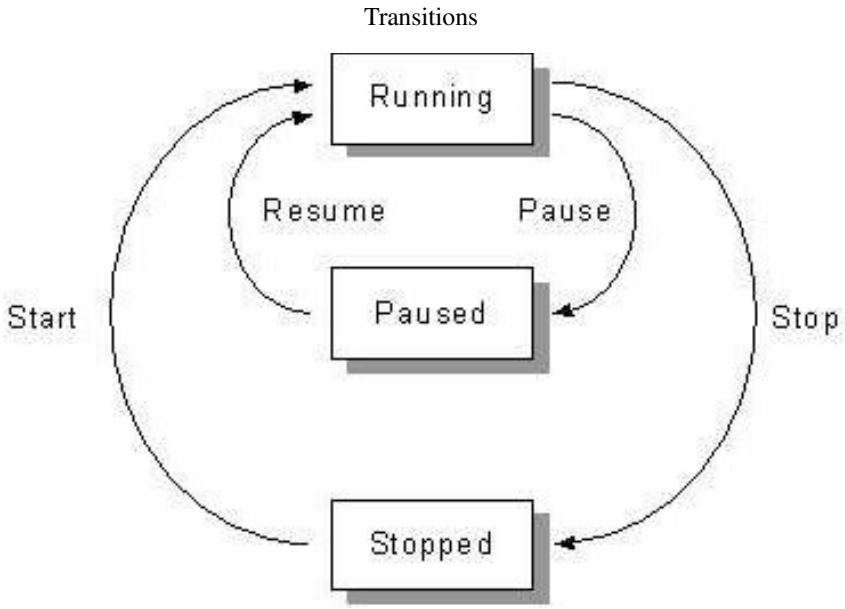


Figure 11: Transitions

6.8.9 Slow Control

The Slow control system is a special front-end equipment or program dedicated to the control of hardware module based on user parameters. It takes advantage of the Online Database and its Hot Link capability. Demand and measured values from slow control system equipment like high voltage power supplies or beam line magnets are stored directly in the ODB.

To control a device it is then enough to modify the demand values in the database. The modified value gets automatically propagated to the slow control system, which in turn uses specific device driver to control the particular hardware. Measured values from the hardware are periodically send back to the ODB to reflect the current status of the sub-system.

The Slow control system is organized in "Classes Driver ". Each Class driver refers to a particular set of functionality of that class i.e. High-Voltage, Temperature, General I/O, Magnet etc. The implementation of the device specific is done in a second stage "Device Driver" while the actual hardware implementation is done in a third layer "Bus

Driver". The current MIDAS distribution already has some device driver for general I/O and commercial High Voltage power supply system (see Supported hardware). The necessary code composing the hardware device driver is kept simple by only requiring a "set channel value" and "read channel value". For the High Voltage class driver, a graphical user interface under Windows or Qt is already available. It can set, load and print high voltages for any devices of that class.

6.8.10 History system

The MIDAS history system is a recording function embedded in the mlogger task. Parallel to its main data logging function of defined channels, the Midas logger can store slow control data and/or periodic events on disk file. Each history entry consists of the time stamp at which the event has occurred and the value[s] of the parameter to be recorded.

The activation of a recording is not controlled by the history function but by the actual equipment (see Frontend code). This permits a higher flexibility of the history system such as dynamic modification of the event structure without restarting the Midas logger. At any given time, data-over-time relation can be queried from the disk file through a Midas utility mhist task or displayed through the mhttpd task.

The history data extraction from the disk file is done using low level file function giving similar result as a standard database mechanism but with faster access time. For instance, a query of a value, which was written once every minute over a period of one week, is performed in a few seconds. For more information see History system, ODB /History Tree.

6.8.11 Alarm System

The Midas alarm mechanism is a built-in feature of the Midas server. It acts upon the description of the required alarm set defined in the Online Database (ODB). Currently the internal alarms supports the following mechanism:

- ODB value over fixed threshold at regular time interval, a pre-defined ODB value will be compared to a fixed value.
- Midas client control During Run state transition, pre-defined Midas client name will be checked if currently present.
- General C-code alarm setting Alarm C function permitting to issue user defined alarm.

The action triggered by the alarm is left to the user through the mean of running a detached script. But basic aalrm report is available such as:

- Logging the alarm message to the experiment log file.

- Sending a "Electronic Log message" (see Electronic Logbook).
- Interrupt data acquisition. For more information see Alarm System, ODB /Alarms Tree.

6.8.12 Electronic Logbook

The Electronic logbook is a feature which provides the experimenter an alternative way of logging his/her own information related to the current experiment. This electronic logbook may supplement or complement the standard paper logbook and in the mean time allow "web publishing" of this information. Indeed the electronic logbook information is accessible from any web browser as long as the mhttpd task is running in the background of the system. For more information see Electronic Logbook, mhttpd task.

Introduction - Top - Quick Start

6.9 Event Builder Functions

Midas supports event building operation through a dedicated mevb task application. Similar to the Midas Frontend application, the mevb task application requires the definition of an equipment structure which describes its mode of operation. The set of parameter for this equipment is limited to:

- Equipment name (appears in the Equipment list).
- Equipment type (should be 0).
- Destination buffer name (SYSTEM if destination event goes to logger).
- Event ID and Trigger mask for the build event (destination event ID).
- Data format (should match the source data format).

Based on the given buffer name provided at the startup time through the *-b buffer_name* argument, the mevb task will scan all the equipments and handle the building of an event based on the identical buffer name found in the equipment list **if the frontend equipment type includes the EQ_EB flag** .

6.9.1 Principle of the Event Builder and related frontend fragment

Possibly in case of multiple frontend, the same "fragment" code may run in the different hardware frontend. In order to prevent to build nFragment different frontend task, the -i index provided at the start of the frontend will replicate the same application image with

the necessary dynamic modification required for the proper Event Building operation. The "-i index" argument will provide the index to be appended to the minimal set of parameter to distinguish the different frontends. These parameters are:

- **frontend_name** : Name of the frontend application.
- **equipment name** : Name of the equipment (from the Equipment structure).
- **event buffer:** Name of the destination buffer (from the Equipment structure).

```
Frontend code:
  /* The frontend name (client name) as seen by other MIDAS clients   */
  char *frontend_name = "ebfe";
  ...
   EQUIPMENT equipment[] = {

      {"Trigger",              /* equipment name */
       1, TRIGGER_ALL,         /* event ID, trigger mask */
       "BUF",                  /* event buffer */
       EQ_POLLED | EQ_EB,       /* equipment type + EQ_EB flag <<<<<< */
       LAM_SOURCE(0, 0xFFFFFF), /* event source crate 0, all stations */
       "MIDAS",                 /* format */
```

Once the frontend is started with *-i 1* , the Midas client name, equipment name and buffer name will be modified.

```
> ebfe -i 1 -D
...
odbedit
[local:midas:S]/Equipment>ls
Trigger01
[local:midas:S]Trigger01>ls -lr
Key name                        Type    #Val  Size  Last Opn Mode Value
---------------------------------------------------------------------------
Trigger01                       DIR
    Common                      DIR
        Event ID                WORD    1     2     18h  0   RWD  1
        Trigger mask            WORD    1     2     18h  0   RWD  65535
        Buffer                  STRING  1     32    18h  0   RWD  BUF01
        Type                    INT     1     4     18h  0   RWD  66
        Source                  INT     1     4     18h  0   RWD  16777215
        Format                  STRING  1     8     18h  0   RWD  MIDAS
        Enabled                 BOOL    1     4     18h  0   RWD  y
        Read on                 INT     1     4     18h  0   RWD  257
        Period                  INT     1     4     18h  0   RWD  500
        Event limit             DOUBLE  1     8     18h  0   RWD  0
        Num subevents           DWORD   1     4     18h  0   RWD  0
        Log history             INT     1     4     18h  0   RWD  0
        Frontend host           STRING  1     32    18h  0   RWD  hostname
        Frontend name           STRING  1     32    18h  0   RWD  ebfe01
        Frontend file name      STRING  1     256   18h  0   RWD  .../eventbuilder/ebfe.c
...
```

Independently of the event ID, each fragment frontend will send its data to the composed event buffer (BUFxx). The event builder task will make up a list of all the equipment belonging to the same event buffer name (BUFxx). If multiple equipments exists in the same frontend, the equipment type (EQ_EB) and the event buffer name will distinguish them.

The Event Builder flowchart below shows a general picture of the event process cycle of the task. The Event Builder runs in polling mode over all the source buffers collected at the begin of run procedure. Once a fragment has been received from all enabled source ("../Settings/Fragment Required y"), an internal event serial number check is performed prior passing all the fragment to the user code. Content of each fragment can be done within the user code for further consistency check.

Event Builder Flowchart.

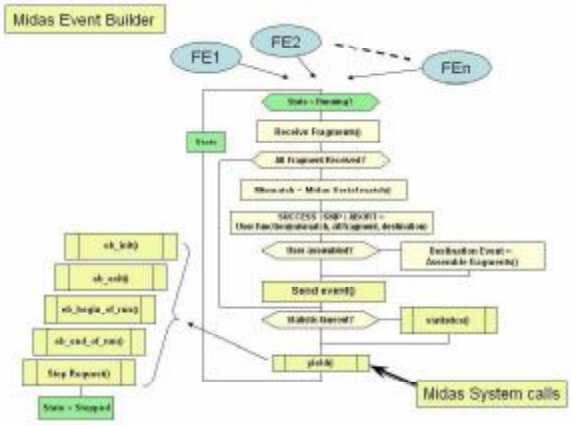


Figure 12: Event Builder Flowchart.

6.9.2 Event builder Tree

The Event builder tree will be created under the Equipment list and will appear as a standard equipment. The sub tree */Common* will contains the specific setting of the equipment while the */Variables* will remain empty. */Settings* will have particular parameter for the Event Builder itself. The **User Field** is an ASCII string passed from the ODB to the eb_begin_of_run() which can be used for steering the event builder.

```
[local:midas:S]EB>ls -lr
Key name                        Type    #Val  Size  Last Opn Mode Value
```

```
------------------------------------------------------------------------------
EB                          DIR
    Common                  DIR
        Event ID            WORD    1     2     5m   0   RWD  1
        Trigger mask        WORD    1     2     5m   0   RWD  0
        Buffer              STRING  1     32    5m   0   RWD  SYSTEM
        Type                INT     1     4     5m   0   RWD  0
        Source              INT     1     4     5m   0   RWD  0
        Format              STRING  1     8     5m   0   RWD  MIDAS
        Enabled             BOOL    1     4     5m   0   RWD  y
        Read on             INT     1     4     5m   0   RWD  0
        Period              INT     1     4     5m   0   RWD  0
        Event limit         DOUBLE  1     8     5m   0   RWD  0
        Num subevents       DWORD   1     4     5m   0   RWD  0
        Log history         INT     1     4     5m   0   RWD  0
        Frontend host       STRING  1     32    5m   0   RWD  hostname
        Frontend name       STRING  1     32    5m   0   RWD  Ebuilder
        Frontend file name  STRING  1     256   5m   0   RWD  c:\...\ebuser.c
    Variables               DIR
    Statistics              DIR
        Events sent         DOUBLE  1     8     3s   0   RWDE 944
        Events per sec.     DOUBLE  1     8     3s   0   RWDE 0
        kBytes per sec.     DOUBLE  1     8     3s   0   RWDE 0
    Settings                DIR
        Number of Fragment  INT     1     4     9s   0   RWD  2
        User build          BOOL    1     4     9s   0   RWD  n
        User Field          STRING  1     64    9s   0   RWD  100
        Fragment Required   BOOL    2     4     9s   0   RWD
                                    [0]             y
                                    [1]             y
```

6.9.3 EB Operation

Using the "eb>" as the cwd for the example, the test procedure is the following: cwd : midas/examples/eventbuilder -> refered as eb>

- Build the mevb task:

```
eb> setenv MIDASSYS /home/midas/midas-1.9.5
eb> make
cc  -g -I/usr/local/include -I../../drivers -DOS_LINUX -Dextname -c ebuser.c
cc  -g -I/usr/local/include -I../../drivers -DOS_LINUX -Dextname -o mevb mevb.c \
          ebuser.o /usr/local/lib/libmidas.a  -lm -lz -lutil -lnsl
cc  -g -I/usr/local/include -I../../drivers -DOS_LINUX -Dextname \
          -c ../../drivers/bus/camacnul.c
cc  -g -I/usr/local/include -I../../drivers -DOS_LINUX -Dextname -o ebfe \
        ebfe.c camacnul.o /usr/local/lib/mfe.o /usr/local/lib/libmidas.a \
        -lm -lz -lutil -lnsl
eb>
```

- Start the following 4 applications in 4 differents windows connecting to a defined experiment. – If no experiment defined yet, set the environment variable MIDAS_DIR to your current directory before spawning the windows.

```
xterm1: eb> ebfe -i 1
xterm2: eb> ebfe -i 2
xterm3: eb> mevb -b BUF
xterm4: eb> odbedit

[local:Default:S]/>ls
System
Programs
Experiment
Logger
Runinfo
Alarms
Equipment
[local:Default:S]/>scl
N[local:midas:S]EB>scl
Name                Host
ebfe01              hostname
ebfe02              hostname
ODBEdit             hostname
Ebuilder            hostname
[local:Default:S]/>
[local:Default:S]/>start now
Starting run #2
```

- The xterm3 (mevb) should display something equivalent to the following, as the print statements are coming from the ebuser code.

- The same procedure can be repeated with the fe1 and fe2 started on remote nodes.

6.10 Internal features

Quick Start - Top - Utilities

This section refers to the Midas built-in capabilities. The following sections describe in more details the essential aspect of each feature starting from the frontend to the Electronic Logbook.

- Run Transition Sequence : Transition Sequence

- Frontend code
  - The Equipment structure : Frontend acquisition characteristics
    - MIDAS event construction : Midas event description
    - YBOS event construction : YBOS event description
    - FIXED event construction :FIXED event description
  - Deferred Transition : Transition postpawning operation
  - Super Event : Short event compaction operation
  - Event Builder Functions : Event Builder operation
- ODB Structure : Online Database Trees
- Hot Link : Notification mechanism
- Alarm System : Alarm scheme
- Slow Control System : Specific Slow Control mechanism
- Electronic Logbook : Essential utility
- Log file : Message, error, report

6.10.1 Run Transition Sequence

The run transition sequence has been modified since Midas version 1.9.5. The new scheme utilize transition sequence level which provides the user a full control of the sequencing of any Midas client.

Midas defines 3 states of Data acquistion: *STOPPED*, *PAUSED*, *RUNNING*

These 3 states require 4 transitions : *TR_START*, *TR_PAUSE* , *TR_RESUME*, *TR_-STOP*

Any Midas client can request notification for run transition. This notification is done by registering to the system for a given transition (cm_register_transition()) by specifying the transition type and the sequencing number (1 to 1000). Multiple registration to a given transition can be requested. This last option permits for example to invoke two callback functions prior and after a given transition such as the start of the logger.

```
my_application.c
  // Callback
  INT before_logger(INT run_number, char *error)
  {
    printf("Initialize ... before the logger gets the Start Transition");
    ...
    return CM_SUCCESS;
  }
```

```
// Callback
INT after_logger(INT run_number, char *error)
{
  printf("Log initial info to file... after logger gets the Start Transition");
  ...
  return CM_SUCCESS;
}

INT main()
{
  ...
  cm_register_transition(TR_START, before_logger, 100);
  cm_register_transition(TR_START, after_logger, 300);
  ...
}
```

By Default the following sequence numbers are used:

- Frontend : TR_START: 500, TR_PAUSE: 500, TR_RESUME: 500,TR_STOP: 500
- Analyzer : TR_START: 500, TR_PAUSE: 500, TR_RESUME: 500,TR_STOP: 500
- Logger : TR_START: 200, TR_PAUSE: 500, TR_RESUME: 500,TR_STOP: 800
- EventBuilder : TR_START: 300, TR_PAUSE: 500, TR_RESUME: 500,TR_-STOP: 700

The sequence number appears into the ODBedit under /System/Clients/

```
[local:midas:S]Clients>ls -lr
Key name                        Type    #Val  Size  Last Opn Mode Value
---------------------------------------------------------------------------
Clients                         DIR
    1832                        DIR     <------------ Frontend 1
        Name                    STRING  1     32    21h  0   R    ebfe01
        Host                    STRING  1     256   21h  0   R    pierre2
        Hardware type           INT     1     4     21h  0   R    42
        Server Port             INT     1     4     21h  0   R    2582
        Transition START        INT     1     4     21h  0   R    500
        Transition STOP         INT     1     4     21h  0   R    500
        Transition PAUSE        INT     1     4     21h  0   R    500
        Transition RESUME       INT     1     4     21h  0   R    500
        RPC                     DIR
            17000               BOOL    1     4     21h  0   R    y
    3872                        DIR     <------------ Frontend 2
        Name                    STRING  1     32    21h  0   R    ebfe02
        Host                    STRING  1     256   21h  0   R    pierre2
        Hardware type           INT     1     4     21h  0   R    42
        Server Port             INT     1     4     21h  0   R    2585
        Transition START        INT     1     4     21h  0   R    500
```

```
    Transition STOP         INT     1     4     21h  0   R    500
    Transition PAUSE        INT     1     4     21h  0   R    500
    Transition RESUME       INT     1     4     21h  0   R    500
    RPC                     DIR
        17000               BOOL    1     4     21h  0   R    y
2220                        DIR     <------------ ODBedit doesn't need transition
    Name                    STRING  1     32    42s  0   R    ODBEdit
    Host                    STRING  1     256   42s  0   R    pierre2
    Hardware type           INT     1     4     42s  0   R    42
    Server Port             INT     1     4     42s  0   R    3429
568                         DIR     <------------ Event Builder
    Name                    STRING  1     32    26s  0   R    Ebuilder
    Host                    STRING  1     256   26s  0   R    pierre2
    Hardware type           INT     1     4     26s  0   R    42
    Server Port             INT     1     4     26s  0   R    3432
    Transition START        INT     1     4     26s  0   R    300
    Transition STOP         INT     1     4     26s  0   R    700
2848                        DIR     <------------ Logger
    Name                    STRING  1     32    5s   0   R    Logger
    Host                    STRING  1     256   5s   0   R    pierre2
    Hardware type           INT     1     4     5s   0   R    42
    Server Port             INT     1     4     5s   0   R    3436
    Transition START        INT     1     4     5s   0   R    200
    Transition STOP         INT     1     4     5s   0   R    800
    Transition PAUSE        INT     1     4     5s   0   R    500
    Transition RESUME       INT     1     4     5s   0   R    500
    RPC                     DIR
        14000               BOOL    1     4     5s   0   R    y
```

The */System/Clients/...* tree reflects the system at a given time. If a permanent change of a client sequence number is required, the system call cm_set_transition_sequence() can be used.

6.10.2 Frontend code

Under MIDAS, experiment hardware is structured into "equipment" which refers to a collection of hardware devices such as: a set of high voltage supplies, one or more crates of digitizing electronics like ADCs and TDCs or a set of scaler. On a software point of view, we keep that same equipment term to refer to the mean of collecting the data related to this "hardware equipment". The data from this equipment is then gathered into an "event" and send to the back-end computer for logging and/or analysis.

The frontend program (image) consists of a system framework contained in mfe.c (hidden from the user) and a user part contained in frontend.c . The hardware access is only apparent in the user code.

Several libraries and drivers exist for various bus systems like CAMAC, VME or RS232. They are located in the drivers directory of the MIDAS distribution. Some libraries consist only of a header file, others of a C file plus a header file. The file names usually refer to the manufacturer abbreviation followed by the model number of

the device. The libraries are continuously expanding to widen Midas support.

ESONE standard routines for CAMAC are supplied and permit to re-use the frontend code among different platforms as well as different CAMAC hardware interface without the need of modification of the code.

The user frontend code consists of several sections described in order below. Example of frontend code can be found under the ../examples/experiment directory:

- **[Global declaration]** Up to the User global section the declarations are system wide and should not be removed.
  - frontend_name This value can be modified to reflect the purpose of the code.
  - frontend_call_loop() Enables the function frontend_loop() to run after every equipment loop.
  - display_period defined in millisecond the time interval between refresh of a frontend status display. The value of zero disable the display. If the frontend is started in a background with the display enabled, the stdout should be redirected to the null device to prevent process to hang.
  - max_event_size specify the maximum size of the expected event in byte.
  - event_buffer_size specify the maximum size of the buffer in byte to be allocated by the system. After these system parameters, the user may add his or her own declarations.

```
// The frontend name (client name) as seen by other MIDAS clients
char *frontend_name = "Sample Frontend";

// The frontend file name, don't change it
char *frontend_file_name = __FILE__;

// frontend_loop is called periodically if this variable is TRUE
BOOL frontend_call_loop = FALSE;

//a frontend status page is displayed with this frequency in ms
INT display_period = 3000;

//maximum event size produced by this frontend
INT max_event_size = 10000;

//buffer size to hold events
INT event_buffer_size = 10*10000;

// Global user section
// number of channels
#define N_ADC  8
#define N_TDC  8
#define N_SCLR 8

CAMAC crate and slots
#define CRATE      0
#define SLOT_C212 23
```

```
#define SLOT_ADC   1
#define SLOT_TDC   2
#define SLOT_SCLR  3
```

- **[Prototype functions]** The first group of prototype(7) declare the pre-defined system functions which should be present. The second group defines the user functions associated to the declared equipments. All the fields are described in detailed in the following section.

```
INT frontend_init();
INT frontend_exit();
INT begin_of_run(INT run_number, char *error);
INT end_of_run(INT run_number, char *error);
INT pause_run(INT run_number, char *error);
INT resume_run(INT run_number, char *error);
INT frontend_loop();

INT read_trigger_event(char *pevent, INT off);
INT read_scaler_event(char *pevent, INT off);
```

  - [Remark] Each equipment has the option to force itself to run at individual transition time see ro_mode . At transition time the system functions begin_of_run(), end_of_run(), pause_run(), resume_run() runs prior to the equipment functions. This gives the system the chance to take basic action on the transition request (Enable/disable LAM) before the equipment runs. The sequence of operation is the following:
    * frontend_init() : Runs once after system initialization, before equipment registration.
    * begin_of_run() : Runs after systerm statistics reset, before any other Equipments at each Begining of Run request.
    * pause_run(): Runs before any other Equipments at each Run Pause request.
    * resume_run(): Runs before any other Equipments at each Run Resume request.
    * end_of_run(): Runs before any other Equipments at each End of Run request.
    * frontend_exit(): Runs once before Slow Control Equipment exit.

- **[Bank definition]** Since the introduction of **ROOT** , the frontend requires to have the definition of the banks in the case you desire to store the raw data in **ROOT** format. This procedure is equivalent to the bank declaration in the analyzer. In the case the format declared is MIDAS, the example below shows the a structured bank and a standard variable length bank declaration for the trigger bank list. The trigger_bank_list[] is declared in the equipment structure (see Eq_example).

```
ADC0_BANK_STR(adc0_bank_str);
BANK_LIST trigger_bank_list[] = {
```

```
   {"ADC0", TID_STRUCT, sizeof(ADC0_BANK), adc0_bank_str},
   {"TDC0", TID_WORD, N_TDC, NULL},
   {""},
};

BANK_LIST scaler_bank_list[] = {
   {"SCLR", TID_DWORD, N_ADC, NULL},
   {""},
};
```

- **[Equipment definition]** See The Equipment structure for further explanation.

```
#undef USE_INT
EQUIPMENT equipment[] = {

  { "Trigger",                                  // equipment name
    {1, 0,                                       // event ID, trigger mask
    "SYSTEM",                                   // event buffer
#ifdef USE_INT
    EQ_INTERRUPT,                               // equipment type
#else
    EQ_POLLED,                                  // equipment type
#endif
    LAM_SOURCE(CRATE, LAM_STATION(SLOT_C212)),  // event source crate 0
    "MIDAS",                                    // format
    TRUE,                                       // enabled
    RO_RUNNING |                                // read only when running
    RO_ODB,                                     // and update ODB
    500,                                        // poll for 500ms
    0,                                          // stop run after this event limit
    0,                                          // number of sub events
    0,                                          // don't log history
    "", "", ""}
    ,
    read_trigger_event,                         // readout routine
    NULL, NULL,
    trigger_bank_list,                          // bank list
    }
  ,
 ...
```

- [frontend_init()] This function run once only at the application startup. Allows hardware checking, loading/setting of global variables, hot-link settings to the ODB etc... In case of CAMAC the standard call can be:

```
    cam_init();                                 // Init CAMAC access
    cam_crate_clear(CRATE);                     // Clear Crate
    cam_crate_zinit(CRATE);                     // Z crate
    cam_inhibit_set(CRATE);                     // Set I crate
    return SUCCESS;
```

- [begin_of_run()] This function is called for every run start transition. Allows to update user parameter, load/setup/clear hardware. At the exit of this function

the acquisition should be armed and ready to test the LAM. In case of CAMAC frontend, the LAM has to be declared to the Crate Controller. The function **cam_lam_enable(CRATE, SLOT_IO)** is then necessary in order to enable the proper LAM source station. The LAM source station has to also be enabled (F26).

The argument **run_number** provides the current run number being started. The argument **error** can be used for returning a message to the system. This string will be logged into the {b midas.log file.

```
// clear units
camc(CRATE, SLOT_C212, 0, 9);
camc(CRATE, SLOT_2249A, 0, 9);
camc(CRATE, SLOT_SC2, 0, 9);
camc(CRATE, SLOT_SC3, 0, 9);

camc(CRATE, SLOT_C212, 0, 26);              // Enable LAM generation

cam_inhibit_clear(CRATE);                   // Remove I

cam_lam_enable(CRATE, SLOT_C212);           // Declare Station to CC as LAM source

// set and clear OR1320 pattern bits
camo(CRATE, SLOT_OR1320, 0, 18, 0x0330);
camo(CRATE, SLOT_OR1320, 0, 21, 0x0663);    // Open run gate, reset latch
return SUCCESS;
```

- [poll_event()] If the equipment definition is **EQ_POLLED** as an acquisition type, the **poll_event()** will be called as often as possible over the corresponding poll time (ex:500ms see The Equipment structure) given by each polling equipment. The code below shows a typical CAMAC LAM polling loop. The **source** corresponds to a bitwise LAM station susceptible to generate LAM for that particular equipement. If the LAM is ORed for several stations and is independent of the equipment, the LAM test can be simplified (see example below)

```
// Trigger event routines ----------------------------------------
INT poll_event(INT source, INT count, BOOL test)
    // Polling routine for events. Returns TRUE if event
    // is available. If test equals TRUE, don't return. The test
    // flag is used to time the polling.
{
  int   i;
  DWORD lam;

  for (i=0 ; i<count ; i++)
    {
      cam_lam_read(LAM_SOURCE_CRATE(source), &lam);
      if (lam & LAM_SOURCE_STATION(source)) // Any of the equipment LAM
   // *** or ***
      if (lam)                              // Any LAM (independent of the equipment)
         if (!test)
          return lam;

  return 0;
```

– **[Remark]** When multiple LAM sources are specified for a given equipment like:

```
LAM_SOURCE(JW_C,  LAM_STATION(GE_N)
                | LAM_STATION(JW_N)),
```

The polling function will pass to the readout function the actual LAM pattern read during the last polling. This pattern is a bitwise LAM station. The content of the **pevent** will be overwritten. This option allows you to determine which of the stations has been the real source of the LAM.

```
INT read_trigger_event(char *pevent, INT off)
{
  DWORD lam;

  lam = *((DWORD *)pevent);

  // check LAM versus MCS station
  // The clear is performed at the end of the readout function
  if (lam & LAM_STATION(JW_N))
  {
   ...
     ...
}
```

- [read_trigger_event()] Event readout function defined in the equipment list. Refer to further section for event composition explanation FIXED event construction , MIDAS event construction , YBOS event construction .

```
// Event readout -------------------------------------------------
INT read_trigger_event(char *pevent, INT off)
{
  WORD *pdata, a;

  // init bank structure
  bk_init(pevent);

  // create ADC bank
  bk_create(pevent, "ADC0", TID_WORD, &pdata);
  ...
}
```

- [pause_run() / resume_run()] These two functions are called respectively upon "Pause" and "Resume" command. Any code relevant to the upcoming run state can be included. Possible commands when CAMAC is involved can be cam_inhibit_set(CRATE) and cam_inhibit_clear(CRATE). The argument **run_number** provides the current run number being paused/resumed. The argument **error** can be used for returning a message to the system. This string will be logged into the midas.log file.

- [end_of_run()] For every "stop run" transition this function is called and provides the opportunity to disable the hardware. In case of CAMAC frontend the LAM should be disabled.

  The argument **run_number** provides the current run number being ended. The argument **error** can be used for returning a message to the system. This string will be logged into the midas.log file.

```
// set and clear OR1320 pattern bits or close run gate.
camo(CRATE, SLOT_OR1320, 0, 18, 0x0CC3);
camo(CRATE, SLOT_OR1320, 0, 21, 0x0990);

camc(CRATE, SLOT_C212, 0, 26);            // Enable LAM generation
cam_lam_disable(CRATE, SLOT_C212);        // disable LAM in crate controller
cam_inhibit_set(CRATE);                   // set crate inhibit
```

- [frontend_exit()] This function runs when the frontend is requested to terminate. Can be used for local statistic collection etc.

6.10.2.1 The Equipment structure To write a frontend program, the user section (frontend.c) has to have an equipment list organized as a structure definition. Here is the structure list for a trigger and scaler equipment from the sample experiment example frontend.c.

```
#undef USE_INT
EQUIPMENT equipment[] = {

  { "Trigger",            // equipment name
    {1, 0,                 // event ID, trigger mask
    "SYSTEM",             // event buffer
#ifdef USE_INT
    EQ_INTERRUPT,         // equipment type
#else
    EQ_POLLED,            // equipment type
#endif
    LAM_SOURCE(0,0xFFFFFF),// event source crate 0, all stations
    "MIDAS",              // format
    TRUE,                 // enabled
    RO_RUNNING |          // read only when running
    RO_ODB,               // and update ODB
    500,                  // poll for 500ms
    0,                    // stop run after this event limit
    0,                    // number of sub events
    0,                    // don't log history
    "", "", ""}
    ,
    read_trigger_event,   // readout routine
    NULL, NULL,
    trigger_bank_list,    // bank list
    }
```

,
. . .

- **["trigger","scaler"]**: Each equipment has to have a unique equipment name defined under a given node. The name will be the reference name of the equipment generating the event.

- **[1, 0]**: Each equipment has to be associated to a unique event ID and a trigger mask. Both the event ID and the trigger mask will be part of the event header of that particular equipment. The trigger mask can be modified dynamically by the readout routine to define a sub-event type on an event-by-event basis. This can be used to mix "physics events" (from a physics trigger) and "calibration events" (from a clock for example) in one run and identify them later. Both parameters are declared as 16bit value. If the Trigger mask is used in a single bit-wise mode, only up to 16 masks are possible.

- **["SYSTEM"]** After composition of an "equipment", the Midas frontend mfe.c takes over the sending of this event to the "system buffer" on the back-end computer. Dedicated buffer can be specified on those lines allowing a secondary stage on the back-end (Event builder to collect and assemble these events coming from different buffers in order to compose a larger event. In this case the events coming from the frontend are called fragment). In this example both events are placed in the same buffer called "SYSTEM" (default).

- **[Remark]** If this field is left empty ("") the readout function associated to that equipment will still be performed, but the actual event won't be sent to the buffer. The positive side-effect of that configuration is to allow that particular equipment to be mirrored in the ODB if the RO_ODB is turned on.

- **[EQ_xxx]** The field specify the type of equipment. It can be of a single type such as EQ_POLLED, EQ_INTERRUPT, EQ_MULTITHREAD, and EQ_SLOW. EQ_POLLED and EQ_MULTITHREAD are similar expect for the polling function which in the case of EQ_MULTITHREAD resides in a separate thread. This new type has been added to take advantage of the multi-core processor and free up CPU for other task than polling.

- [EQ_POLLED] In this mode, the name of the routine performing the trigger check function is defaulted to poll_event(). As polling consists of checking a variable for a true condition, if the loop would be infinite, the frontend would not be able to respond to any network commands. Therefore the loop count is determined when the frontend starts so that it returns after a given time-out if no event is available. This time-out is usually in the order of 500 milliseconds. This flag is mainly used for data acquisition based on a "LAM".

```
EQUIPMENT equipment[] = {

  { "Trigger",            // equipment name   ...
    500,                  // poll for 500ms
  ...
```

- [EQ_INTERRUPT] For this mode, Midas requires complete configuration and control of the interrupt source. This is provided by an interrupt configuration routine interrupt_configure() that has to be coded by the user in the user section of the frontend code. A pointer to this routine is passed to the system instead of the polling routine. The interrupt configuration routine has the following declaration:

```
INT interrupt_configure(INT cmd, INT source [], PTYPE adr)
{
  switch(cmd)
    {
    case CMD_INTERRUPT_ENABLE:
      cam_interrupt_enable();
      break;
    case CMD_INTERRUPT_DISABLE:
      cam_interrupt_disable();
      break;
    case CMD_INTERRUPT_ATTACH:
      cam_interrupt_attach((void (*)())adr);
      break;
    case CMD_INTERRUPT_DETACH:
      cam_interrupt_detach();
      break;

  return CM_SUCCESS;
```

- [EQ_PERIODIC] In this mode the function associated to this equipment is called periodically. No hardware requirements is necessary to trigger the readout function. The "poll" field in the equipment declaration is in this case used for periodicity.
- [EQ_MULTITHREAD] This new equipment type is valid since version 2.0. It implements the multi-threading capability within the frontend code. The polling is performed within a separate thread and uses the ring buffer functions rb_xxx for inter-thread communication.
- [EQ_SLOW] Declare the equipment as a Slow Control equipment. This will enable the call to the **idle** function part of the class driver.
- [EQ_MANUAL_TRIG] This flag enables the equipment to be triggered by remote procedure call (RPC). If present, the web interface will provide a button for that action.
- [EQ_FRAGMENTED] This flag enables large events (beyond Midas configuration limit) to be handled by the system. This flag requires to have a valid

max_event_size_frag variable defined in the user frontend code (frontend.c). The max_event_size variable is used as fragment size in this case. This option is meant to be used in experiments where the event rate is not an issue but the size of the data needs to be extremely large. In any selected case, when the equipment is required to run, a declared function is called doing the actual user required operation. Under the four commands listed above, the user has to implement the adequate hardware operation performing the requested action. In **drivers** examples can be found on such an interrupt code. See source code such as hyt1331.c, ces8210.c.

  - CMD_INTERRUPT_ENABLE: to enable an interrupt
  - CMD_INTERRUPT_DISABLE: to disable an interrupt
  - CMD_INTERRUPT_INSTALL: to install an interrupt callback routine at address adr.
  - CMD_INTERRUPT_DEINSTALL: to de-install an interrupt.

- [EQ_EB] This flag identifies the equipment as a **fragment event** and should be ored with the EQ_POLLED in order to be identified by the Event_Builder.

- [LAM_SOURCE(0,0xFFFFFF)] This parameter is a bit-wise representation of the 24 CAMAC slots which may raise the LAM. It defines which CAMAC slot is allowed to trigger the call to the readout routine. (See read_trigger_event()).

- **["MIDAS"]** This line specifies the data format used for generating the event. The following options are possible: MIDAS, YBOS and FIXED. The format has to agree with the way the event is composed in the user read-out routine. It tells the system how to interpret an event when it is copied to the ODB or displayed in a user-readable form.

MIDAS and YBOS or FIXED and YBOS data format can be mixed at the frontend level, but the data logger (mlogger) is not able to handle this format diversity on a event-by-event basis. In practice a given experiment should keep the data format identical throughout the equipment definition.

- [TRUE] "enable" switch for the equipment. Only when enable (TRUE) the related equipment is active.

- [RO_RUNNING] Specify when the read-out of an event should be occurring (transition state) or be enabled (state). Following options are possible:

RO_RUNNING	Read on state "running"
RO_STOPPED	Read on state "stopped"
RO_PAUSED	Read on state "paused"
RO_BOR	Read after begin-of-run
RO_EOR	Read before end-of-run
RO_PAUSE	Read when run gets paused
RO_RESUME	Read when run gets resumed
RO_TRANSITIONS	Read on all transitions
RO_ALWAYS	Read independently of the states and force a read for all transitions.
RO_ODB	Equipment event mirrored into ODB under variables

These flags can be combined with the logical OR operator. Trigger events in the above example are read out only when running while scaler events is read out when running and additionally on all transitions. A special flag RO_ODB tells the system to copy the event to the /Equipment/<equipment name>/Variables ODB tree once every ten seconds for diagnostic. Later on, the event content can then be displayed with ODBEdit.

- [500] Time interval for Periodic equipment (EQ_PERIODIC) or time out value in case of EQ_POLLING (unit in millisecond).
- [0 (stop after...)] Specify the number of events to be taken prior to forcing an End-Of-Run transition. The value 0 disables this option.
- [0 (Super Event)] Enable the Super event capability. Specify the maximum number of events in the Super event.
- [0 (History system)] Enable the MIDAS history system for that equipment. The value (positive in seconds) indicates the time interval for generating the event to be available for history logging by the mlogger task if running.
- ["","","",""] Reserved field for system. Should be present and remain empty.
- [read_trigger_event()] User read-out routine declaration (could be any name). Every time the frontend is initialized, it copies the equipment settings to the ODB under /Equipment/<equipment name>/Common. A hot-link to that ODB tree is created allowing some of the settings to be changed during run-time. Modification of "Enabled" flag, RO_xxx flags, "period" and "event limit" from the ODB is immediately reflected into the frontend which will act upon them. This function has to be present in the frontend code and will be called for every trigger under one of the two conditions:
  - [In polling mode] The poll_event has detected a trigger request while polling on a trigger source.
  - [In interrupt mode] An interrupt source pre-defined through the interrupt_configuration has occurred.

– [Remark 1] The first argument of the readout function provides the pointer to the newly constructed event and points to the first valid location for storing the data.

– [Remark 2] The content of the memory location pointed by **pevent** prior to its uses in the readout function contains the LAM source bitwise register. This feature can be exploited in order to identify which slot has triggered the readout when multiple LAM has been assigned to the same readout function. **Example:**

```
... in the equipment declaration
    ...
    LAM_SOURCE(JW_C,  LAM_STATION(GE_N) | LAM_STATION(JW_N)), // event source
    ...
    "", "", "",
    event_dispatcher,   // readout routine
...

INT event_dispatcher(char *pevent)
{
  DWORD lam, dword;
  INT   size=0;
  EQUIPMENT      *eq;

  // the *pevent contains the LAM pattern returned from poll_event
  //  The value can be used to dispatch to the proper LAM function

  // !!!! ONLY one of the LAM is processed in the loop !!!!
  lam = *((DWORD *)pevent);

  // check LAM versus MCS station
  if (lam & LAM_STATION(JW_N))
  {
    ...
    // read MCS event
    size = read_mcs_event(pevent);
    ...

  else if (lam & LAM_STATION(GE_N))
  {
    ...
    // read GE event
    size = read_ge_event(pevent);
    ...

  return size;
```

– [Remark 3] In the example above, the Midas Event Header contains the same Event ID as the Trigger mask for both LAM. The event serial number will be incremented by one for every call to event_dispatcher() as long as the returned size is non-zero.

– [Remark 4] The return value should represent the number of bytes collected in this function. If the returned value is set to zero, The event will be dismissed and the serial number to that event will be decremented by one.

6.10.2.2 FIXED event construction The FIXED format is the simplest event format. The event length is fixed and is mapped to a C structure that is filled by the readout routine. Since the standard MIDAS analyzer cannot work with this format, it is only recommended for experiment, which uses its own analyzer and wants to avoid the overhead of a bank structure. For fixed events, the structure has to be defined twice: Once for the compiler in form of a C structure and once for the ODB in form of an ASCII representation. The ASCII string is supplied to the system as the "init string" in the equipment list.

Following statements would define a fixed event with two ADC and TDC values:

```
typedef struct {
  int adc0;
  int adc1;
  int tdc0;
  int tdc1;
  TRIGGER_EVENT;
char *trigger_event_str[] = {
"adc0 = INT : 0",
"adc1 = INT : 0",
"tdc0 = INT : 0",
"tdc1 = INT : 0",
  ASUM_BANK;
```

The **trigger_event_str** has to be defined before the equipment list and a reference to it has to be placed in the equipment list like:

```
{
...
  read_trigger_event, // readout routine
  poll_trigger_event, // polling routine
  trigger_event_str,  // init string
 ,
```

The readout routine could then look like this, where the <...> statements have to be filled with the appropriate code accessing the hardware:

```
INT read_trigger_event(char *pevent)
{
TRIGGER_EVENT *ptrg;

  ptrg = (TRIGGER_EVENT *) pevent;
  ptrg->adc0 = <...>;
  ptrg->adc1 = <...>;
  ptrg->tdc0 = <...>;
  ptrg->tdc1 = <...>;

  return sizeof(TRIGGER_EVENT);
```

6.10.3 MIDAS event construction

The MIDAS event format is a variable length event format. It uses "banks" as subsets of an event. A bank is composed of a bank header followed by the data. The bank header itself is made of 3 fields i.e: bank name (4 char max), bank type, bank length. Usually a bank contains an array of values that logically belong together. For example, an experiment can generate an ADC bank, a TDC bank and a bank with trigger information. The length of a bank can vary from one event to another due to zero suppression from the hardware. Besides the variable data length support of the bank structure, another main advantage is the possibility for the analyzer to add more (calculated) banks during the analysis process to the event in process. After the first analysis stage, the event can contain additionally to the raw ADC bank a bank with calibrated ADC values called CADC bank for example. In this CADC bank the raw ADC values could be offset or gain corrected.

MIDAS banks are created in the frontend readout code with calls to the MIDAS library. Following routines exist:

- bk_init() , bk_init32() Initializes a bank structure in an event.
- bk_create() Creates a bank with a given name (exactly four characters)
- bk_close() Closes a bank previously opened with bk_create().
- bk_locate() Locates a bank within an event by its name.
- bk_iterate() Returns bank and data pointers to each bank in the event.
- bk_list() Constructs a string with all the banks' names in the event.
- bk_size() Returns the size in bytes of all banks including the bank headers in an event. The following code composes a event containing two ADC and two TDC values, the <...> statements have to be filled with specific code accessing the hardware:

```
INT read_trigger_event(char *pevent)
{
INT *pdata;

  bk_init(pevent);

  bk_create(pevent, "ADC0", TID_INT, &pdata);
  *pdata++ = <ADC0>
  *pdata++ = <ADC1>
  bk_close(pevent, pdata);

  bk_create(pevent, "TDC0", TID_INT, &pdata);
  *pdata++ = <TDC0>
  *pdata++ = <TDC1>
  bk_close(pevent, pdata);

  return bk_size(pevent);
```

Upon normal completion, the readout routine returns the event size in bytes. If the event is not valid, the routine can return zero. In this case no event is sent to the back-end. This can be used to implement a software event filter (sometimes called "third level trigger").

```
INT read_trigger_event(char *pevent)
{
WORD *pdata, a;

  // init bank structure
  bk_init(pevent);

  // create ADC bank
  bk_create(pevent, "ADC0", TID_WORD, &pdata);

  // read ADC bank
  for (a=0 ; a<8 ; a++)
    cami(1, 1, a, 0, pdata++);

  bk_close(pevent, pdata);

  // create TDC bank
  bk_create(pevent, "TDC0", TID_WORD, &pdata);

  // read TDC bank
  for (a=0 ; a<8 ; a++)
    cami(1, 2, a, 0, pdata++);

  bk_close(pevent, pdata);

  return bk_size(pevent);
```

6.10.4 YBOS event construction

The YBOS event format is also a bank format used in other DAQ systems. The advantage of using this format is the fact that recorded data can be analyzed with pre-existing analyzers understanding YBOS format. The disadvantage is that it has a slightly larger overhead than the MIDAS format and it supports fewer bank types. An introduction to YBOS can be found under:

YBOS

The scheme of bank creation is exactly the same as for MIDAS events, only the routines are named differently. The YBOS format is double word oriented i.e. all incrementation are done in 4 bytes steps. Following routines exist:

- ybk_init() Initializes a bank structure in an event.
- ybk_create() Creates a bank with a given name (exactly four characters)

- ybk_close() Closes a bank previously opened with ybk_create().
- ybk_size() Returns the size in bytes of all banks including the bank headers in an event.

The following code creates an ADC0 bank in YBOS format:

```
INT read_trigger_event(char *pevent)
{
  DWORD i;
  DWORD *pbkdat;

  ybk_init((DWORD *) pevent);

  // collect user hardware data
  ybk_create((DWORD *)pevent, "ADC0", I4_BKTYPE, (DWORD *)(&pbkdat));
  for (i=0 ; i<8 ; i++)
    *pbkdat++ = i & 0xFFF;
  ybk_close((DWORD *)pevent, pbkdat);

  ybk_create((DWORD *)pevent, "TDC0", I2_BKTYPE, (DWORD *)(&pbkdat));
  for (i=0 ; i<8 ; i++)
    *((WORD *)pbkdat)++ = (WORD)(0x10+i) & 0xFFF;
  ybk_close((DWORD *) pevent, pbkdat);

  ybk_create((DWORD *)pevent, "SIMU", I2_BKTYPE, (DWORD *)(&pbkdat));
  for (i=0 ; i<9 ; i++)
    *((WORD *)pbkdat)++ = (WORD) (0x20+i) & 0xFFF;
  ybk_close((DWORD *) pevent, I2_BKTYPE, pbkdat);

  return (ybk_size((DWORD *)pevent));
```

6.10.5 Deferred Transition

This option permits the user to postpone any transition issued by any requester until some condition are satisfied. As examples:

- It may not be advised to pause or stop a run until let say some hardware has turned off a particular valve.
- The start of the acquisition system is postponed until the beam rate has been stable for a given period of time.
- While active, a particular acquisition system should not be interrupted until the "cycle" is complete.

In these examples, any application having access to the state of the hardware can register to be a "transition Deferred" client. It will then catch any transition request and postpone the trigger of such transition until *condition* is satisfied. The Deferred_Transition requires 3 steps for setup:

- Register the deferred transition.

```
//-- Frontend Init
INT frontend_init()
{
  INT    status, index, size;
  BOOL   found=FALSE;

  // register for deferred transition
  cm_register_deferred_transition(TR_STOP, wait_end_cycle);
  cm_register_deferred_transition(TR_PAUSE, wait_end_cycle);
  ...
```

- Provide callback function to serve the deferred transition

```
//-- Deferred transition callback
BOOL wait_end_cycle(int transition, BOOL first)
{
  if (first)
  {
    transition_PS_requested = TRUE;
    return FALSE;


  if (end_of_mcs_cycle)
  {
    transition_PS_requested = FALSE;
    end_of_mcs_cycle = FALSE;
    return TRUE;

  else
    return FALSE;
  ...
```

- Implement the condition code

```
... In this case at the end of the readout function...
  ...
INT read_mcs_event(char *pevent, INT offset)
{
  ...

  if (transition_PS_requested)
  {
    // Prevent to get new MCS by skipping re_arm_cycle and GE by GE_DISABLE LAM
    cam_lam_disable(JW_C,JW_N);
    cam_lam_disable(GE_C,GE_N);
    cam_lam_clear(JW_C,JW_N);
    cam_lam_clear(GE_C,GE_N);
    camc(GE_C,GE_N,0,GE_DISABLE);
    end_of_mcs_cycle = TRUE;

  re_arm_cycle();
  return bk_size(pevent);
```

In the example above the frontend code register for PAUSE and STOP. The second argument of the cm_register *wait_end_cycle* is the declaration of the callback function. The callback function will be called as soon as the transition is requested and will provide the Boolean flag first to be TRUE. By setting the *transition_PS_requested* , the user will have the acknowledgment of the transition request. By returning FALSE from the callback you will prevent the transition to occur. As soon as the user condition is satisfied (end_of_mcs_cycle = TRUE), the return code in the callback will be set to TRUE and the requested transition will be issued. The Deferred transition shows up in the ODB under **/runinfo/Requested transition** and will contain the transition code (see State Codes & Transition Codes). When the system is in deferred state, an ODBedit override command can be issued to **force** the transition to happen. eg: odbedit> stop now, odbedit> start now . This overide will do the transition function regarless of the state of the hardware involved.

6.10.6 Super Event

The Super Event is an option implemented in the frontend code in order to reduce the amount of data to be transfered to the backend by removing the bank header for each event constructed. In other words, when an equipment readout in either *MIDAS* or *YBOS* format (bank format) is complete, the event is composed of the bank header followed by the data section. The overhead in bytes of the bank structure is 16 bytes for bk_init(), 20 bytes for bk_init32() and ybk_init(). If the data section size is close to the number above, the data transfer as well as the data storage has an non-negligible overhead. To address this problem, the equipment can be setup to generate a so called **Super Event** which is an event composed of the initial standard bank header for the first event of the super event and up to **number of sub event** maximum successive data section before the closing of the bank.

To demonstrate the use of it, let's see the following example:

- Define equipment to be able to generate *Super* Event

```
{ "GE",                 // equipment name
    2, 0x0002,            // event ID, trigger mask
    "SYSTEM",             // event buffer
#ifdef USE_INT
    EQ_INTERRUPT,         // equipment type
#else
    EQ_POLLED,            // equipment type
#endif
    LAM_SOURCE(GE_C, LAM_STATION(GE_N)),      // event source
    "MIDAS",              // format
    TRUE,                 // enabled
    RO_RUNNING,           // read only when running
    200,                  // poll for 200ms
```

```
  0,                    // stop run after this event limit
  1000,                 // -----> number of sub event <-----  enable Super event
  0,                    // don't log history
  "", "", "",
  read_ge_event,        // readout routine
   ,
  ...
```

- Setup the readout function for Super Event collection.

```
//-- Event readout
// Global and fixed -- Expect NWORDS 16bits data readout per sub-event
#define NWORDS 3

INT read_ge_event(char *pevent, INT offset)
{
  static WORD *pdata;

  // Super event structure
  if (offset == 0)
  {
    // FIRST event of the Super event
    bk_init(pevent);
    bk_create(pevent, "GERM", TID_WORD, &pdata);

  else if (offset == -1)
  {
    // close the Super event if offset is -1
    bk_close(pevent, pdata);

    // End of Super Event
    return bk_size(pevent);


  // read GE sub event (ADC)
  cam16i(GE_C, GE_N, 0, GE_READ, pdata++);
  cam16i(GE_C, GE_N, 1, GE_READ, pdata++);
  cam16i(GE_C, GE_N, 2, GE_READ, pdata++);

  // clear hardware
  re_arm_ge();

  if (offset == 0)
  {
    // Compute the proper event length on the FIRST event in the Super Event
    // NWORDS correspond to the !! NWORDS WORD above !!
    // sizeof(BANK_HEADER) + sizeof(BANK) will make the 16 bytes header
    // sizeof(WORD) is defined by the TID_WORD in bk_create()

    return NWORDS * sizeof(WORD) + sizeof(BANK_HEADER) + sizeof(BANK);

  else
    // Return the data section size only
    // sizeof(WORD) is defined by the TID_WORD in bk_create()

    return NWORDS * sizeof(WORD);
```

The encoded description of the data section is left to the user. If the number of words per sub-event is fixed (NWORD), the sub-event extraction is simple. In the case of variable sub-event length, it is necessary to tag the first or the last word of each sub-event. The content of the sub-event is essentially the responsibility of the user.

- [Remark 1] The backend analyzer will have to be informed by the user on the content structure of the data section of the event as no particular tagging is applied to the **Super Event** by the Midas transfer mechanism.
- [Remark 2] If the **Super Event** is composed in a remote equipment running a different *Endian* mode than the backend processor, it would be necessary to insure the data type consistency throughout the **Super Event** in order to guarantee the proper byte swapping of the data content.
- [Remark 3] The event rate in the equipment statistic will indicates the rate of sub-events.

6.10.7 Slow Control System

Instead of talking directly to each other, frontend and control programs exchange information through the ODB. Each slow control equipment gets a corresponding ODB tree under /Equipment. This tree contains variables needed to control the equipment as well as variables measured by the equipment. In case of a high voltage equipment this is a Demand array which contains voltages to be set, a Measured array which contains read back voltages and a Current array which contains the current drawn from each channel. To change the voltage of a channel, a control program writes to the Demand array the desired value. This array is connected to the high voltage frontend via a ODB hot-link. Each time it gets modified, the frontend receives a notification and sets the new value. In the other direction the frontend continuously reads the voltage and current values from all channels and updates the according ODB arrays if there has been a significant change. This design has a possible inconvenience due to the fact that ODB is the key element of that control. Any failure or corruption of the database can result in wrong driver control. Therefore it is not recommended to use this system to control systems that need redundancy for safety purposes. On the other hand this system has several advantages:

- The control program does not need any knowledge of the frontend, it only talks to the ODB.
- The control variables only exist at one place that guarantees consistency among all clients.
- Basic control can be done through ODBEdit without the need of a special control program.

- A special control program can be tested without having a frontend running.
- In case of n frontend and m control programs, only n+m network connections are needed instead of n∗m connection for point-to-point connections. Since all slow control values are contained in the ODB, they get automatically dumped to the logging channels. The slow control frontend uses the same framework as the normal frontend and behaves similar in many respects. They also create periodic events that contain the slow control variables and are logged together with trigger and scaler events. The only difference is that a routine is called periodically from the framework that has the task to read channels and to update the ODB. To access slow control hardware, a two-layer driver concept is used. The upper layer is a "class driver", which establishes the connection to the ODB variables and contains high level functionality like channel limits, ramping etc. It uses a "device driver" to access the channels. These drivers implement only very simple commands like "set channel" and "read channel". The device drivers themselves can use bus drivers like RS232 or GPIB to control the actual device.

Class driver, Device and Bus driver in the slow control system

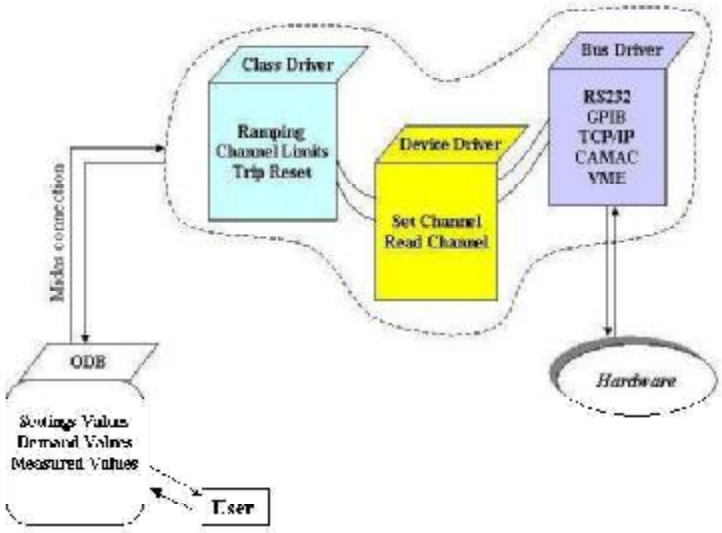


Figure 13: Class driver, Device and Bus driver in the slow control system

The separation into class and device drivers has the advantage that it is very easy to add new devices, because only the simple device driver needs to be written. All higher

functionality is inherited from the class driver. The device driver can implement richer functionality, depending on the hardware. For some high voltage devices there is a current read-back for example. This is usually reflected by additional variables in the ODB, i.e. a Current array. Frontend equipment uses exactly one class driver, but a class driver can use more than one device driver. This makes it possible to control several high voltage devices for example with one frontend in one equipment. The number of channels for each device driver is defined in the slow control frontend. Several equipment with different class drivers can be defined in a single frontend.

```
Key name                        Type    #Val  Size  Last Opn Mode Value
--------------------------------------------------------------------------
Epics                           DIR
    Settings                    DIR
        Channels                DIR
            Epics               INT     1     4     25h  0   RWD  3
        Devices                 DIR
            Epics               DIR
                Channel name    STRING  10    32    25h  0   RWD
                                        [0]             GPS:VAR1
                                        [1]             GPS:VAR2
                                        [2]             GPS:VAR3
        Names                   STRING  10    32    17h  1   RWD
                                        [0]             Current
                                        [1]             Voltage
                                        [2]             Watchdog
        Update Threshold Measure FLOAT  10    4     17h  0   RWD
                                        [0]             2
                                        [1]             2
                                        [2]             2
    Common                      DIR
        Event ID                WORD    1     2     17h  0   RWD  3
        Trigger mask            WORD    1     2     17h  0   RWD  0
        Buffer                  STRING  1     32    17h  0   RWD  SYSTEM
        Type                    INT     1     4     17h  0   RWD  4
        Source                  INT     1     4     17h  0   RWD  0
        Format                  STRING  1     8     17h  0   RWD  FIXED
        Enabled                 BOOL    1     4     17h  0   RWD  y
        Read on                 INT     1     4     17h  0   RWD  121
        Period                  INT     1     4     17h  0   RWD  60000
        Event limit             DOUBLE  1     8     17h  0   RWD  0
        Num subevents           DWORD   1     4     17h  0   RWD  0
        Log history             INT     1     4     17h  0   RWD  1
        Frontend host           STRING  1     32    17h  0   RWD  hostname
        Frontend name           STRING  1     32    17h  0   RWD  Epics
        Frontend file name      STRING  1     256   17h  0   RWD  feepic.c
    Variables                   DIR
        Demand                  FLOAT   10    4     0s   1   RWD
                                        [0]             1.56
                                        [1]             120
                                        [2]             87
        Measured                FLOAT   10    4     2s   0   RWD
                                        [0]             1.56
                                        [1]             120
                                        [2]             87
    Statistics                  DIR
```

```
Events sent             DOUBLE  1     8     17h  0   RWDE 26
Events per sec.         DOUBLE  1     8     17h  0   RWDE 0
kBytes per sec.         DOUBLE  1     8     17h  0   RWDE 0
```

6.10.8 Electronic Logbook

The Electronic logbook is an alternative way of recording experiment information. This is implemented through the Midas web server mhttpd task (see Elog page). The definition of the options can be found in the ODB data base under ODB /Elog Tree.

6.10.9 Log file

Midas provides a general log file **midas.log** for recording system and user messages across the different components of the data acquisition clients. The location of this file is dependent on the mode of installation of the system.

1. [without ODB /Logger Tree] In this case the location is defined by either the MIDAS_DIR environment (see Environment variables) or the definition of the experiment in the **exptab** file (see Experiment_Definition). In both cases the log file will be in the experiment specific directory.
2. [with /Logger Tree] The **midas.log** will be sitting into the defined directory specified by **Data Dir** .

midas.log file contains system and user messages generated by any application connected to the given experiment.

The MIDAS Macros definition provides a list of possible type of messages.

```
Fri Mar 24 10:48:40 2000 [CHAOS] Run 8362 started
Fri Mar 24 10:48:40 2000 [Logger] Run #8362 started
Fri Mar 24 10:55:04 2000 [Lazy_Tape] cni-043[10] (cp:383.6s) /dev/nst0/run08360.ybs 849.896MB file N
Fri Mar 24 11:24:03 2000 [MStatus] Program MStatus on host umelba started
Fri Mar 24 11:24:03 2000 [MStatus] Program MStatus on host umelba stopped
Fri Mar 24 11:27:02 2000 [Logger] stopping run after having received 1200000 events
Fri Mar 24 11:27:03 2000 [CHAOS] Run 8362 stopped
Fri Mar 24 11:27:03 2000 [SUSIYBOS] saving info in run log
Fri Mar 24 11:27:03 2000 [Logger] Run #8362 stopped
Fri Mar 24 11:27:13 2000 [Logger] starting new run
Fri Mar 24 11:27:14 2000 [CHAOS] Run 8363 started
Fri Mar 24 11:27:14 2000 [CHAOS] odb_access_file -I- /Equipment/kos_trigger/Dump not found
Fri Mar 24 11:27:14 2000 [Logger] Run #8363 started
Fri Mar 24 11:33:47 2000 [Lazy_Tape] cni-043[11] (cp:391.8s) /dev/nst0/run08361.ybs 850.209MB file N
Fri Mar 24 11:42:35 2000 [CHAOS] Run 8363 stopped
Fri Mar 24 11:42:40 2000 [SUSIYBOS] saving info in run log
```

```
Fri Mar 24 11:42:41 2000 [ODBEdit] Run #8363 stopped
Fri Mar 24 12:19:57 2000 [MChart] client [umelba.Triumf.CA]MChart failed watchdog test after 10 sec
Fri Mar 24 12:19:57 2000 [MChart] Program MChart on host koslx0 stopped
```

Quick Start - Top - Utilities

6.11 Introduction

New Documented Features - Top - Components

... A few words...

Acquiring, collecting and analyzing data is the essence of mankind to satisfy his urge for understanding natural phenomena by comparing "real" events to his own symbolic representation. These fundamental steps paved human evolution and in the world of science they have been the keys to major steps forward in our understanding of nature. Until the last couple of decade's -when "Silicium" was still underground, the PPP protocol (Paper, Pencil and Patience) was the basic tool for this "unique" task. With the development of the "Central Processing Unit", data acquisition using computers wired to dedicated hardware instrumentation became available. This has allowed scientists to sit back and turn their minds towards finding solutions to problems such as "How do I analyze all these data?" Since the last decade or so when "connectivity" appeared to be a powerful word, the data acquisition system had to adapt itself to that new vocabulary.

Based on this sudden new technology, several successful systems using decentralization of information have been developed. But the task is not simple! If the hardware is available, implementing a true distributed intelligence environment for a particular application requires that each node have full knowledge of the capability of all the other nodes. Complexity rises quickly and generalization of such systems is tough. Recently more pragmatic approaches emerged from all this, suggesting that central database information on a system may be more adequate, especially since processing and networking speed are not a "real" concern these days. MIDAS and its predecessor HIX may be counted part of the precursor packages in the field.

The old question: "How do we analyze all these data?" still remains and may have been the driving force behind this evolution :-).

6.11.1 What is Midas?

The Maximum Integrated Data Acquisition System (MIDAS) is a general-purpose system for event based data acquisition in small and medium scale physics experiments. It has been developed at the Paul Scherrer Institute (Switzerland) and at TRIUMF (Canada) between 1993 and 2000 (Release of Version 1.8.0). Presently ongoing development are more focused on the interfacing capability of the Midas package to external applications such as ROOT for data analysis (see MIDAS Analyzer).

Midas is based on a modular networking capability and a central database system.

MIDAS consists of a C library and several applications. They run on many different operating systems such as UNIX like, Windows NT, VxWorks, VMS and MS-DOS. While the system is already in use in several laboratories, the development continues with addition of new features and tools. Current development involves RTLinux for either dedicated frontend or composite frontend and backend system.

For the newest status, check the MIDAS home page: Switzerland , Canada

6.11.2 What can MIDAS do for you?

MIDAS has been designed for small and medium experiments. It can be used in distributed environments where one or more frontends are connected to the backend via Ethernet. The frontend might be an embedded system like a VME CPU running VxWorks or a PC running Windows NT or Linux. Data rates around 1MB/sec through standard Ethernet and 6.1MB/sec over Fast Ethernet can be achieved.

For small experiments and test setups the front-end program can run on the back-end computer thus eliminating the need of network transfer, presuming that the back-end computer has direct access to the hardware. Device drivers for common PC-CAMAC interfaces have been written for Windows NT and Linux. Drivers for PC-VME interfaces are commercially available for Windows NT.

For data analysis, users can write a complete analyzer or use the standard MIDAS analyzer which uses HBOOK routines for histogramming and PAW for histogram display.

The MIDAS package contains also a slow control system which can be used to control high voltage supplies, temperature control units etc. The slow control system is fully integrated in the main data acquisition and act as a front-end with particular built-in control mechanism. Slow control values can be written together with event data to tape.

New Documented Features - Top - Components

6.12 mhttpd task

Utilities - Top - Data format

mhttpd is the Midas Web Server. It provides Midas DAQ control through the web using any web browser.

This daemon application has to run in order to allow the user to access from a Web browser any Midas experiment running on a given host. Full monitoring and "Almost" full control of a particular experiment can be achieved through this Midas Web server. The color coding is green for present/enabled, red for missing/disabled, yellow for inactive. It is important to note the refresh of the page is not "event driven" but is controlled by a timer (see **Config-** button). This mean the information at any given time may reflect the experiment state of up to n second in the paste, where n is the timer setting of the refresh parameter. Its basic functionality are:

- Run control (start/stop).
- Frontend up-to-date status and statistics display.
- Logger up-to-date status and statistics display.
- Lazylogger up-to-date status and statistics display.
- Current connected client listing.
- Slow control data display.
- Basic access to ODB.
- Graphical history data display.
- Electronic LogBook recording/retrival messages
- Alarm monitoring/control
- ... and more ...

Each section is further described below:

- Start page - Run control page
- ODB page - Online Database manipulation (equivalent to ODBedit)
- Equipment page (Frontend info)
- CNAF page (CAMAC access page)
- Message page (Message Log)
- Elog page (Electronic Log)
  - Internal Elog (Internal)
  - External Elog (External)
- Program page (Program control)
- History page (History display)
- Alarm page (Alarm control)
- Custom page (User defined Web page)

mhttpd requires as argument the TCP/IP port number in order to listen to the web based request.

- **Arguments**

- [-h] : help
- [-p port] : port number, no default, should be 8081 for **Example** .
- [-D] : start program as a daemon

- **Usage**

```
>mhttpd -p 8081 -D
```

- Description Once the connection to a given experiment is established, the main Midas status page is displayed with the current ODB information related to this experiment. The page is sub-divised in several sections:

-[Experiment/Date] Current Experiment, current date.

-[Action/Pages buttons] Run control button, Page switch button. At any web page level within the Midas Web page the main status page can be invoked with the <status> button.

- [Start... button] Depending on the run state, a single or the two first buttons will be showing the possible action (Start/Pause/Resume/Stop) (see Start page).
- [ODB button] Online DataBase access. Depending on the security, R/W access can be granted to operated on any ODB field (see ODB page).
- [CNAF button] If one of the equipment is a CAMAC frontend, it is possible to issue CAMAC command through this button. In this case the frontend is acting as a RPC CAMAC server for the request (see CNAF page).
- [Messages button] Shows the n last entries of the Midas system message log. The last entry is always present in the status page (see below) (see Message page).
- [Elog button] Electronic Log book. Permit to record permanently (file) comments/messages composed by the user (see Elog page).
- [Alarms button] Display current Alarm setting for the entire experiment. The activation of an alarm has to be done through ODB under the **/Alarms** tree (See Alarm System)
- [Program button] Display current program (midas application) status. Each program has a specific information record associated to it. This record is tagged as a hyperlink in the listing (see Program page).
- [History button] Display History graphs of pre-defined variables. The history setting has to be done through ODB under the **/History** (see History system , History page).
- [Config button] Allows to change the page refresh rate.

- [Help button] Help and link to the main Midas web pages.
- [User button(s)] If the user define a new tree in ODB named **Script** than any sub-tree name will appear as a button of that name. Each sub-tree (/Script/<button name>/) should contain at least one string key being the script command to be executed. Further keys will be passed as
  - **Arguments** to the script. Midas Symbolic link are permitted.
  - **Example** : The **Example** below defines a script names doit with 2 **Arguments** (run# device) which will be invoked when the button <doit> is pressed.

```
odbedit
mkdir Script
cd Script
mkdir doit
cd doit
create string cmd
ln "/runinfo/run number" run
create string dest
set dest /dev/hda
```

[Version >= 1.8.3 Alias Hyperlink] This line will be present on the status page only if the ODB tree /**Alias**. The distinction for spawning a secondary frame with the link request is done by default. For forcing the link in the current frame, add the terminal charater "&" at the end of the link name.

- **Example** : The **Example** will create a shortcut to the defined location in the ODB.

```
odbedit
ls
create key Alias
cd Alias
ln /Equipment/Trigger/Common "Trig Setting"
ln /Analyzer/Output "Analyzer"

create key "Alias new window"                    <-- Version < 1.8.3
cd "Alias new window"
ln /equipment/Scalers/Variables "Scalers Var"

or
cd Alias
ln /Equipment/Trigger/Common "Trig Setting&"     <-- Version >= 1.8.3
```

- [General info] Current run number, state, General Enable flag for Alarm, Auto restart flag Condition of mlogger.
- [Equipment listing] Equipment name (see Equipment page), host on which its running, Statistics for that current run, analyzed percentage by the "analyzer" (The numbers are valid only if the name of the analyser is "Analyzer").

- [Logger listing] Logger list. Multiple logger channel can be active (single application). The hyperlink "0" will bring you to the odb tree /Logger/channels/0/Settings. This section is present only when the Midas application mlogger task is running.
- [Lazylogger listing] Lazylogger list. Multiple lazy application can be active. This section is present only when the Midas application lazylogger task is running.
- [Last system message] Display a single line containing the last system message received at the time of the last display refresh.
- [Current client listing] List of the current active Midas application with the hostname on which their running.

Midas Web server

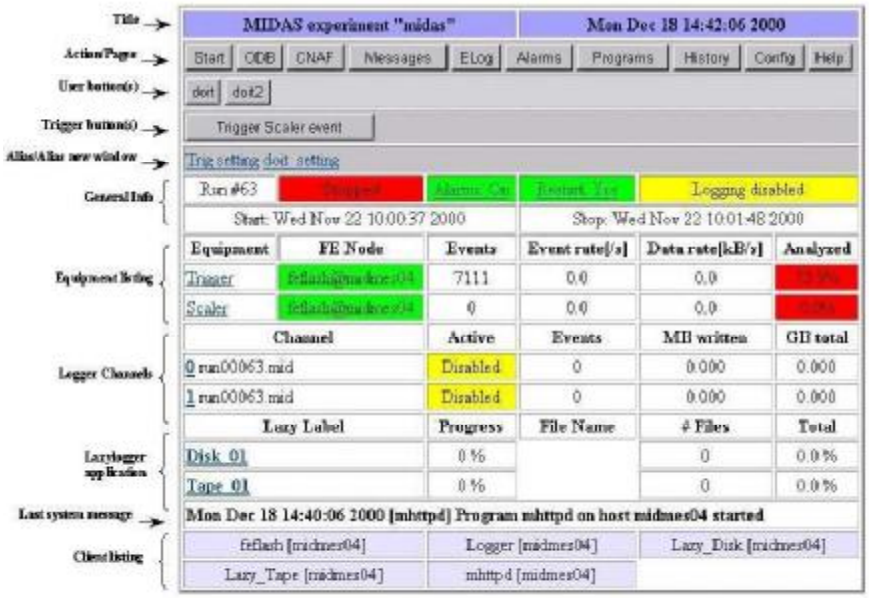


Figure 14: Midas Web server

6.12.1 Start page

Once the **Start** button is pressed, you will be prompt for experiment specific parameters before starting the run. The minimum set of parameter is the run number, it will be incremented by one relative to the last value from the status page. In the case you have defined the ODB tree **/Experiment/Edit on Start** all the parameters sitting in this directory will be displayed for possible modification. The **Ok** button will proceed to the start of the run. The **Cancel** will abort the start procedure and return you to the status page.

Start run request page. In this case the user has multiple run parameters defined under "/Experiment/Edit on Start"

MIDAS experiment "e614"	Tue Dec 19 09:50:16 2000
Start new run	
Run number	895
Comment	Test, -150 mv th
Write Data	y
Exp type	3 mod test
Operators	SCW RP
Sc 1 HV (volts)	2300
Sc 2 HV (volts)	1800
GAS type	Ar 25 Iso 75
U1 HV (volts)	-2000
V1 HV (volts)	-2000
U2 HV (volts)	-2000
V2 HV (volts)	-1750
U3 HV (volts)	-2000
V3 HV (volts)	-2000
Preamp (mV)	4200
Start Cancel	

Figure 15: Start run request page.

The title of each field is taken from the ODB key name it self. In the case this label has a poor meaning and extra explanation is required, you can do so by creating a new ODB tree under experiment **Parameter Comments/** . Then by creating a string entry named as the one in **Edit** on Start- you can place the extra information relative to that key (html tags accepted).

This "parameter comment" option is available and visible **ONLY** under the midas web page, the **odbedit start** command will not display this extra information.

```
[local:midas:S]/Experiment>ls -lr
Key name                        Type    #Val  Size  Last Opn Mode Value
--------------------------------------------------------------------------------
Experiment                      DIR
    Name                        STRING  1     32    17s  0   RWD  midas
    Edit on Start               DIR
        Write data              BOOL    1     4     16m  0   RWD  y
        enable                  BOOL    1     4     16m  0   RWD  n
        nchannels               INT     1     4     16m  0   RWD  0
        dwelling time (ns)      INT     1     4     16m  0   RWD  0
    Parameter Comments          DIR
        Write Data              STRING  1     64    44m  0   RWD  Enable logging
        enable                  STRING  1     64    7m   0   RWD  Scaler for expt B1 only
        nchannels               STRING  1     64    14m  0   RWD  <i>maximum 1024</i>
        dwelling time (ns)      STRING  1     64    8m   0   RWD  <b>Check hardware now</b>

[local:midas:S]Edit on Start>ls -l
Key name                        Type    #Val  Size  Last Opn Mode Value
--------------------------------------------------------------------------------
Write Data                      LINK    1     19    50m  0   RWD  /logger/Write data
enable                          LINK    1     12    22m  0   RWD  /sis/enable
number of channels              LINK    1     15    22m  0   RWD  /sis/nchannels
dwelling time (ns)              LINK    1     24    12m  0   RWD  /sis/dwelling time (ns)
```

Start run request page. Extra comment on the run condition is displayed below each entry.

Figure 16: Start run request page.

6.12.2 ODB page

The ODB page shows the ODB root tree at first. Clicking on the hyperlink will walk you to the requested ODB field. The **Example** below show the sequence for changing the variable "PA" under the /equipment/PA/Settings/Channels ODB directory. A possible shortcut

If the ODB is Write protected, a first window will request the web password.

ODB page access.

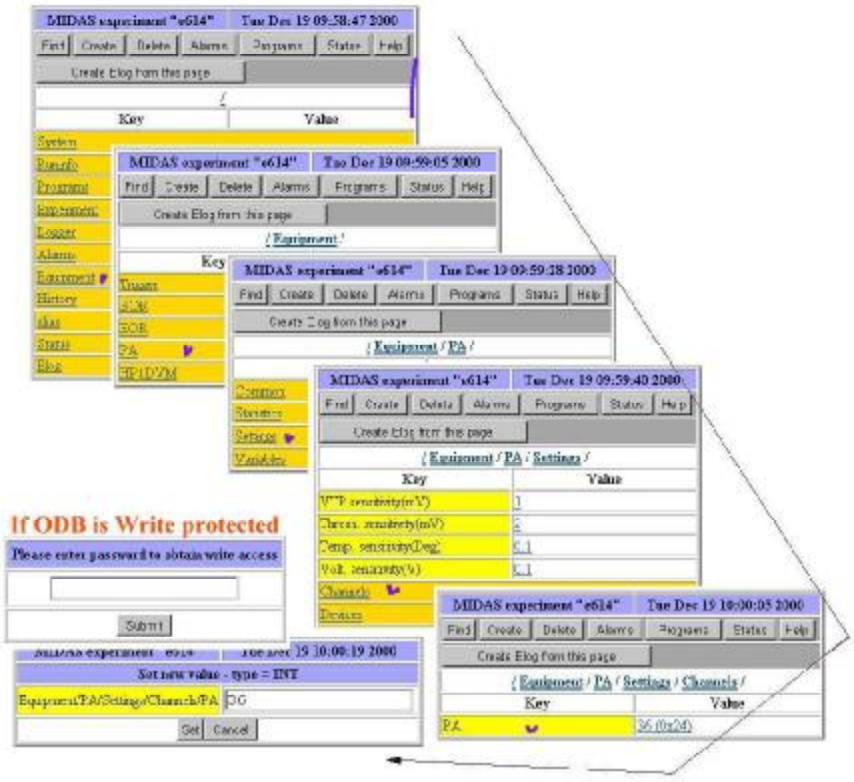


Figure 17: ODB page access.

6.12.3 Equipment page

The equipment names are linked to their respective **/Variables** sub-tree. This permit to access as a shortcut the current values of the equipment. In the case the equipment is a slow control equipment, the parameters list may be hyperlinked for parameter modification. This option is possible only if the parameter names have a particular name syntax (see History system).

Slow control page.

MIDAS experiment "e614" | Mon Dec 18 14:21:54 2000

ODB | Status | Help

Equipment: **PA**

Groups: All **Crate0** Crate1

Names	D_VTp	M_VTp	D_Thres	M_ThresA	M_ThresB	D_TP	M_TP	Temp	Voltage+	Voltage-
SI_0	0	0	0	0	0	n	n	51	-0.018	-0.006
SI_1	1850	1852	1011	-1002	-998	n	n	31.3	5.061	-5.103
SI_2	1793	1793	1017	-1002	-999	n	n	33.8	5.099	-5.112
SI_3	1775	1774	1023	-1001	-1000	n	n	33.5	5.067	-5.093
SI_4	1852	1852	1017	-1003	-999	n	n	34.9	5.076	-5.104
SI_5	1800	1800	1014	-1004	-1000	n	n	38.5	5.055	-5.108
SI_6	1786	1785	1011	-1001	-1000	n	n	40.4	5.066	-5.098
SI_7	1798	1798	1011	-1004	-1000	n	n	37.3	5.083	-5.097
SI_8	1795	1795	1018	-1002	-1002	n	n	32	5.073	-5.092
SI_9	1801	1801	1016	-1001	-1002	n	n	35.1	5.09	-5.104
SI_10	1797	1798	1033	-1001	-1000	n	n	34.7	5.065	-5.104
SI_11	1795	1796	1019	-1000	-1002	n	n	31.3	5.057	-5.102
SI_12	1797	0	1013	0	0	n	n	0	-0.022	-0.006
SI_13	1798	1798	1016	-1002	-1000	n	n	34.3	5.067	-5.102
SI_14	1793	1793	1016	-1000	-1000	n	n	32.4	5.07	-5.095
SI_15	1799	1800	1015	-1000	-1001	n	n	28.9	5.068	-5.092
SI_16	1782	1783	1007	-1002	-1001	n	n	37.7	5.058	-5.099
SI_17	1798	1798	1011	-1001	-999	n	n	33.3	5.104	-5.094
SI_18	1796	1796	1017	-1001	-1002	n	n	30.6	5.078	-5.103
SI_19	1798	1797	1009	-1000	-1001	n	n	34.7	5.07	-5.106
SI_20	1803	1803	1014	-1002	-1000	n	n	37.6	5.066	-5.11
SI_21	1799	1799	1010	-1000	-1002	n	n	38.7	5.056	-5.11
SI_22	1805	1805	1015	-1000	-1001	n	n	33.1	5.066	-5.114
SI_23	1793	1793	1019	-1000	-1001	n	n	31.2	5.055	-5.096
SI_24	1789	1788	1018	-1000	-1002	n	n	38.1	5.047	-5.105

Figure 18: Slow control page.

6.12.4 CNAF page

If one of the active equipment is a CAMAC based data collector, it will be possible to remotely access CAMAC through this web based CAMAC page. The status of the connection is displayed in the top right hand side corner of the window.

CAMAC command pages.

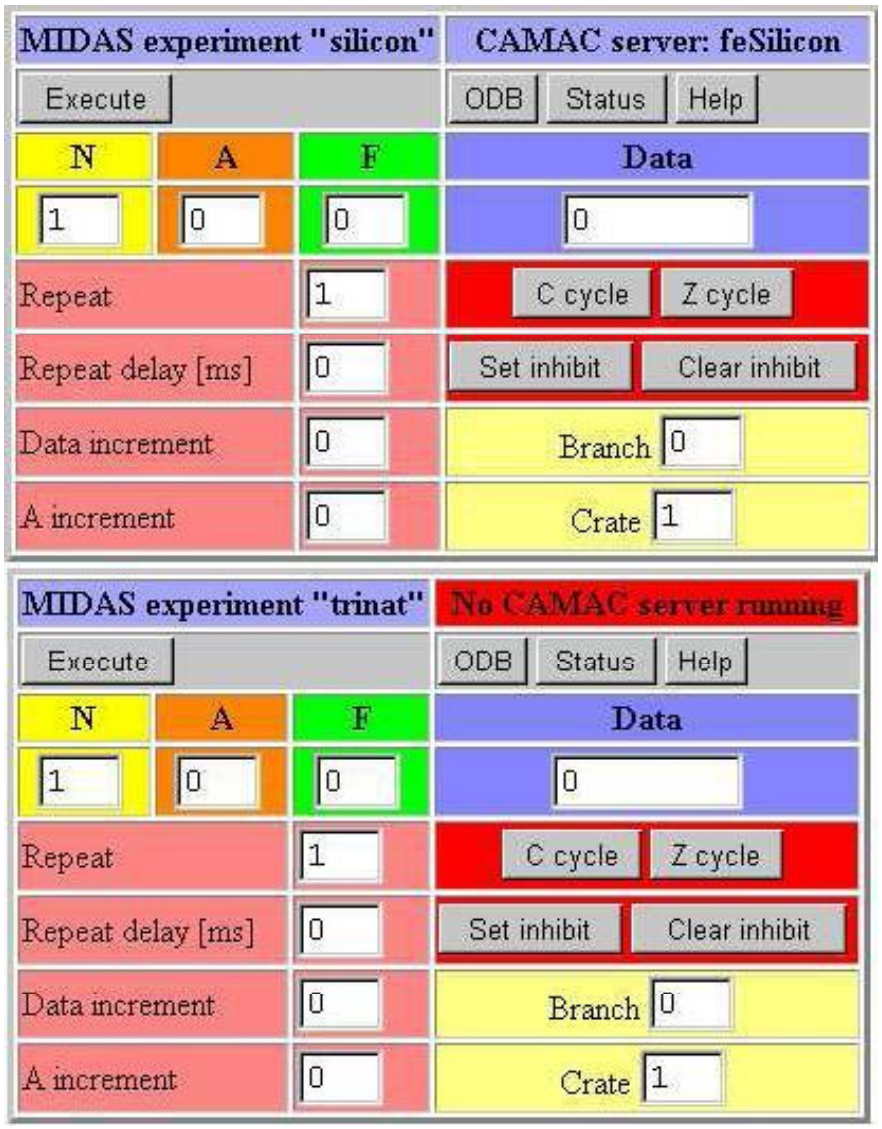


Figure 19: CAMAC command pages.

6.12.5 Message page

This page display by block of 100 lines the content of the Midas System log file starting with the most recent messages. The Midas log file resides in the directory defined by the experiment.

Message page.

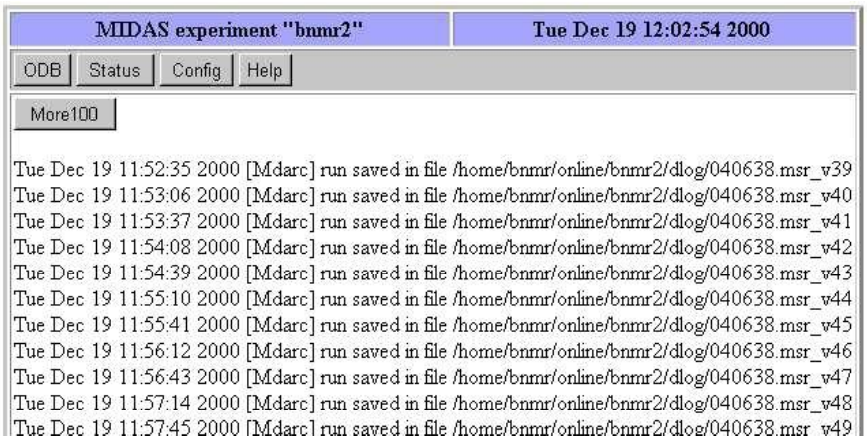


Figure 20: Message page.

6.12.6 Elog page

The ELOG page provides access to an electronic logbook. This tool can replace the experimental logbook for daily entries. The main advantage of Elog over paper logbook is the possiblity to access it remotely and provide a general knowledge of the experiment. In the other hand, Elog is not limited strictly to experiments and worldwide Elog implementation can be found on the internet.

Since version 2.0.0, Elog comes in two flavors, i.e Internal Elog where the Elog is built in the mhttpd Midas web interface, or External Elog where the Elog runs independently from the experiment and mhttpd as well. While the internal doesn't requires any setup, the latter requires a proper Elog installation which is fully described on the `Elog` web site. The External Elog implementation requires to have a dedicated entry in the ODB following the code below. It requires also to have the package Elog previously installed and properly configured. Once the ODB entry is existant, the internal ELOG is disabled.

6.12.6.1 Internal Elog By default the mhttpd provides the internal Elog. The entry destination directory is established by the logger key in ODB (see Elog_Dir) The Electronic Log page shows the most recent Log message recorded in the system. The top buttons allows you to either Create/Edit/Reply/Query/Show a message

main Elog page.

Figure 21: main Elog page.

The format of the message log can be written in HTML format.

HTML Elog message.

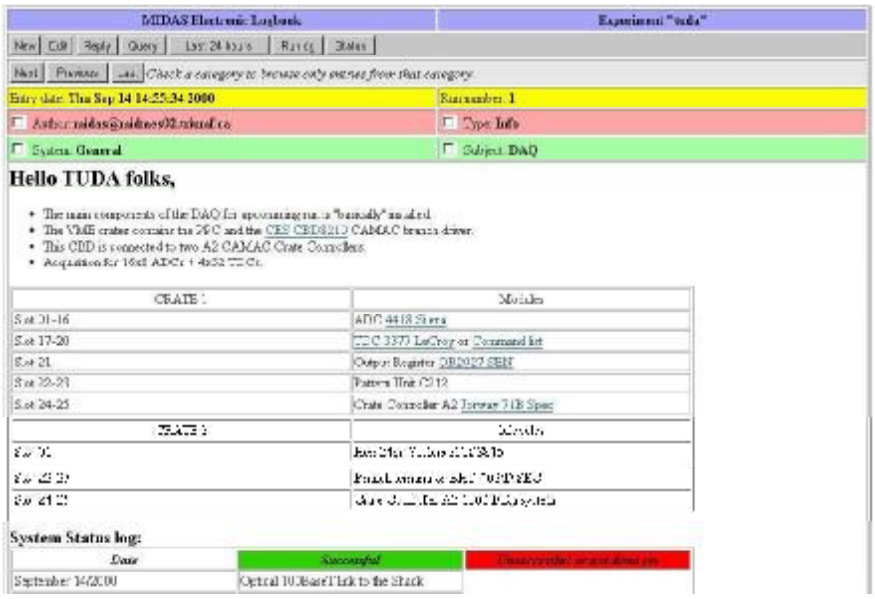


Figure 22: HTML Elog message.

- A feature of the Elog entry page is the **Shift Check** button, this permit for the experimenter in shift to go through a check list and record his findings in the Elog system. The check list is user defined and can be found in the ODB under **/Elog**

HTML Elog message.

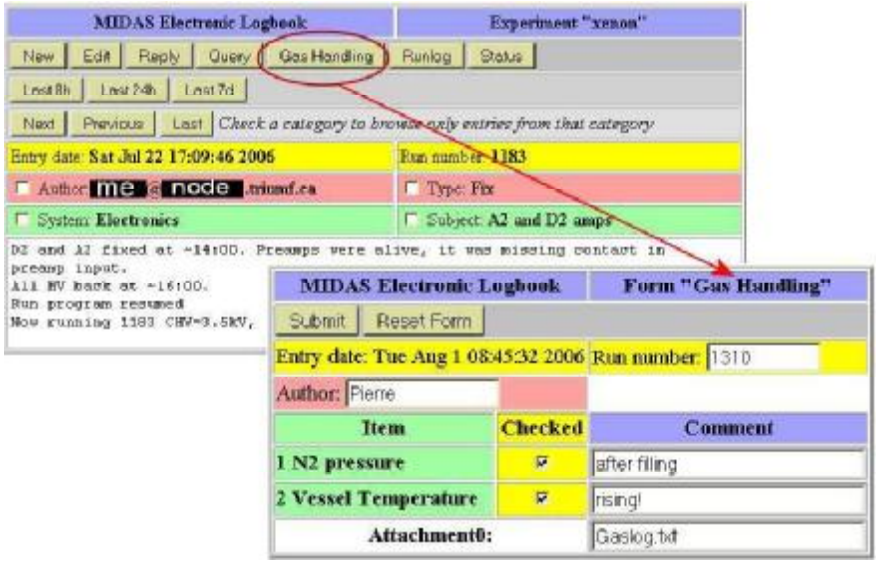


Figure 23: HTML Elog message.

- The code below generates the above screen. The key *Gas Handling* contains all the information for a given form. There is no limit to the number of entries. By specifying an entry with the name *Attachment0*,*Attachment1*,... and filling it with a fix file name, its content will be attached to the Elog entry for every shift report.

```
[local:myexpt:Running]/>cd /Elog/
[local:myexpt:Running]/Elog>mkdir Forms
[local:myexpt:Running]/Elog>cd Forms/
[local:myexpt:Running]Forms>mkdir "Gas Handling"
[local:myexpt:Running]Forms>cd "Gas Handling"
[local:myexpt:Running]Gas Handling>create string "N2 Pressure"
String length [32]:
[local:myexpt:Running]Gas Handling>create string "Vessel Temperature"
String length [32]:
[local:myexpt:Running]Gas Handling>ls
N2 pressure
Vessel Temperature
[local:myexpt:Running]Gas Handling>
[local:xenon:Running]Gas Handling>create string Attachment0
String length [32]: 64
[local:xenon:Running]Gas Handling>set Attachment0 Gaslog.txt
```

- The **runlog** button display the content of the file **runlog.txt** which is expected to be in the data directory specified by the ODB key **/Logger/Data Dir**. Regardless of its content, it will be displayed in the web page. Its common uses is to **append**

lines after every run. The task appending this run information can be any of the midas application. **Example** is available in the *examples/experiment/analyzer.c* which at each end-of-run (EOR) will write to the runlog.txt some statistical informations.

Elog page, Runlog display.

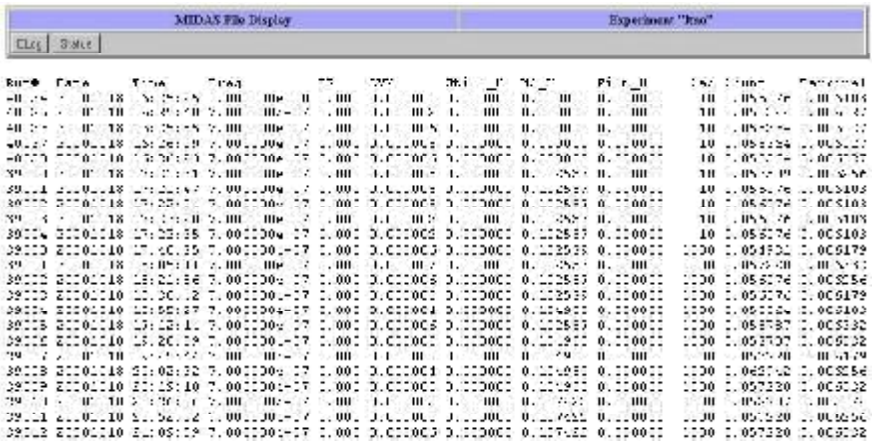


Figure 24: Elog page, Runlog display.

- When composing a new entry into the Elog, several fields are available to specify the nature of the message i.e: Author, Type, System, Subject. Under Type and System a pulldown menu provides multiple category. These categories are user definable through the odb under the tree **/Elog/Types**, **/Elog/Systems**. The number of category is fixed to 20 maximum but any remaining field can be left empty.

Elog page, New Elog entry form.

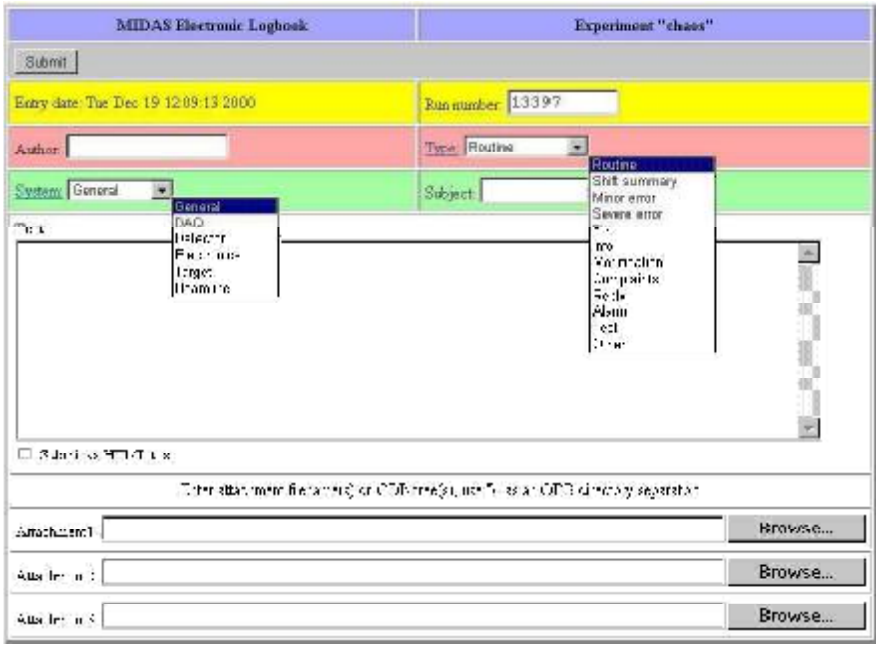


Figure 25: Elog page, New Elog entry form.

6.12.6.2 External Elog The advantage of using the external Elog over the built-in version is its felxibility. This package is used worldwide and impovement is constantly added. A full features documentations and standalone installation can be found at the Elog web site.

It's installation requires requires several steps described below.

- Download the Elog package from the mentioned web site.
  - Windows, Linux, Mac version can be found there. Simple installation procedures are also described. Its installation can be done at the system level or at the user level. The Elog can service multiple Electronic logbooks in parallel and therefore an extra entry in its configuration file can provide specific experimental elog in a similar fashion as the internal one.
  - You need to take note of several consideration for its installation. You need to determine several locations for the different files that elog deals with.
    - elog resource directory (ex: /elog_installation_dir where elog is installed)

    * logbook directory (ex: /myexpt/logbook where the pwd and elog entries are stored). The pwd file uses encryption for the user password.

– As this Elog installation is tailored towards an experiment, a restriction applies i.e: Ensure that the mhttpd and elog applications shares at least the same file system. This means that either both applications runs on the same machine or a nsf mount provides file sharing.
    * You need to now the node and ports for both application. As mhttpd, elogd requires a port number for communication through the web (ex: NodeA:mhttpd -p 8080, NodeB:elogd -p 8081.

1. copy the default midas/src/elogd.cfg from the midas distrbution to your operating directory.
2. modify the elogd.cfg to reflect your configuration

```
# This is a simple elogd configuration file to work with Midas
# $Id: mhttpd.dox 3317 2006-09-06 04:01:31Z amaudruz $

[global]
; port under which elogd should run
port = 8081
; password file, created under 'logbook dir'
password file = elog.pwd
; directory under which elog was installed (themes etc.)
resource dir = /elog_installation_dir
; directory where the password file will end up
logbook dir = /myexpt/logbook
; anyone can create it's own account
self register = 1
; URL under which elogd is accessible
url = http://ladd00.triumf.ca:8081
; the "main" tab will bring you back to mhttpd
main tab = Xenon
; this is the URL of mhttpd which must run on a different port
main tab url = http://NodeA:8080
; only needed for email notifications
smtp host = your.smtp.host
; Define one logbook for online use. Severl logbooks can be defined here
[MyOnline]
; directory where the logfiles will be written to
Data dir = /myexpt/logbook
Comment = My MIDAS Experiment Electronic Logbook
; mimic old mhttpd behaviour
Attributes = Run number, Author, Type, System, Subject
Options Type = Routine, Shift Summary, Minor Error, Severe Error, Fix, Question, Info
Options System = General, DAQ, Detector, Electronics, Target, Beamline
Extendable Options = Type, System
; This substitution will enter the current run number
Preset Run number = $shell(odbedit -e myexpt -h NodeA -d Runinfo -c 'ls -v \"run numb
Preset Author = $long_name
Required Attributes = Type, Subject
; Run number and Author cannot be changed
Locked Attributes = Run number, Author
Page Title = ELOG - $subject
Reverse sort = 1
Quick filter = Date, Type, Author
```

```
; Don't send any emails
Suppress email to users = 1
```

3. start the elog daemon. **-x** is for the shell substitution of the command *Preset Run number = $shell(...* The argument invokes the odbedit remotely if needed to retrieve the current run number. You will have to ensure the proper path to the odbedit and the proper -e, -h arguments for the experiment and host. You may want to verify this command from the console.

```
NodeB:~>/installation_elog_dir/elogd -c elogd.cfg -x
```

4. start the mhttpd at its correct port and possibly in the daemon form.

```
NodeA:~>mhttpd -p 8080 -D
```

5. At this point the Elog from the Midas web page is accessing the internal Elog. To activate the external Elog, include in the ODB two entries such as:

```
NodeX:> odbedit -e myexpt -h NodeA
[NodeX:myexpt:Running]/>cd elog
[NodeX:myexpt:Running]/Elog>create string Url
String length [32]: 64
[NodeX:myexpt:Running]/Elog>set Url http://NodeB:8081/MyOnline
[NodeX:myexpt:Running]
[NodeX:myexpt:Running]/Elog>create string "Logbook Dir"
String length [32]: 64
[NodeX:myexpt:Running]/Elog>set "Logbook Dir" /myexpt/logbook

[NodeX:myexpt:Running]/Elog>ls
Logbook Dir                    /home/myexpt/ElogBook
Url                            http://NodeB:8081/MyOnline
```

6. Confirm proper operation of the external Elog by creating an entry. You will be prompt for a username and password. Click on New registration. Full control of these features are described in the Elog documentation.
7. Stop and restart the Elogd in the background.

```
NodeB:~>/installation_elog_dir/elogd -c elogd.cfg -x -D
```

8. In the event you had previous entry under the internal elog, you can convert the internal to external using the elconv tool.

```
NodeB:~> cp internal/elog_logbook/*.log /myexpt/logbook/.
NodeB:~> cd /myexpt/logbook
NodeB:~> /installation_elog_dir/elconv
```

6.12.7 Program page

This page present the current active list of the task attached to the given experiment. On the right hand side a dedicated button allows to stop the program which is equivalent to the ODBedit command **odbedit**> sh <task name> .

The task name hyperlink pops a new window pointing to the ODB section related to that program. The ODB structure for each program permit to apply alarm on the task presence condition and automatic spawning at either the begining or the end of a run.

Program page.

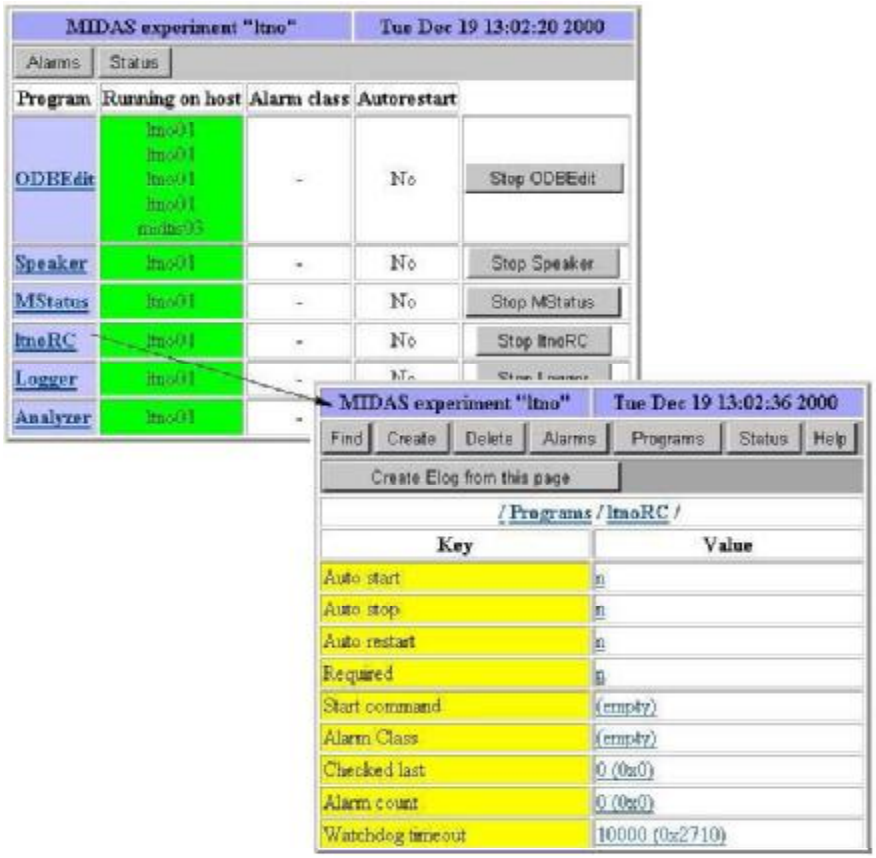


Figure 26: Program page.

6.12.8 History page

This page reflects the History system settings (CVS r1.271). It lists on the top of the page the possible group names containing a list of panels defined in the ODB. Next

a serie of buttons defines the time scale of the graph with predefined time window, "<<","<" "+" "-" ">" ">>" buttons permit the shifting of the graph in the time direction. Other buttons will allow graph resizing, Elog attachment creation, configuration of the panel and custom time frame graph display. By default a single group is created "Default" containing the trigger rate for the "Trigger" equipment.

The configuration options for a given panel consists in:

- Zooming capability, run markers, logarithmic scale.
- Data query in time.
- Time scale in date format.
- Web based page creation ("new" button) for up to 10 history channels per page.

History page.

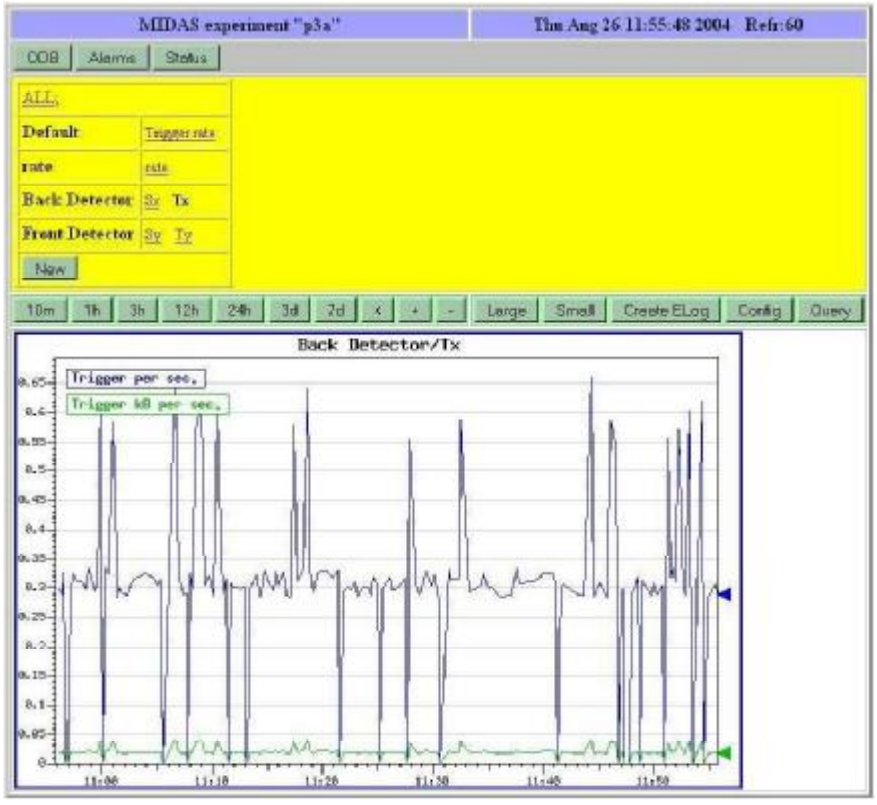


Figure 27: History page.

History channel selection Page.

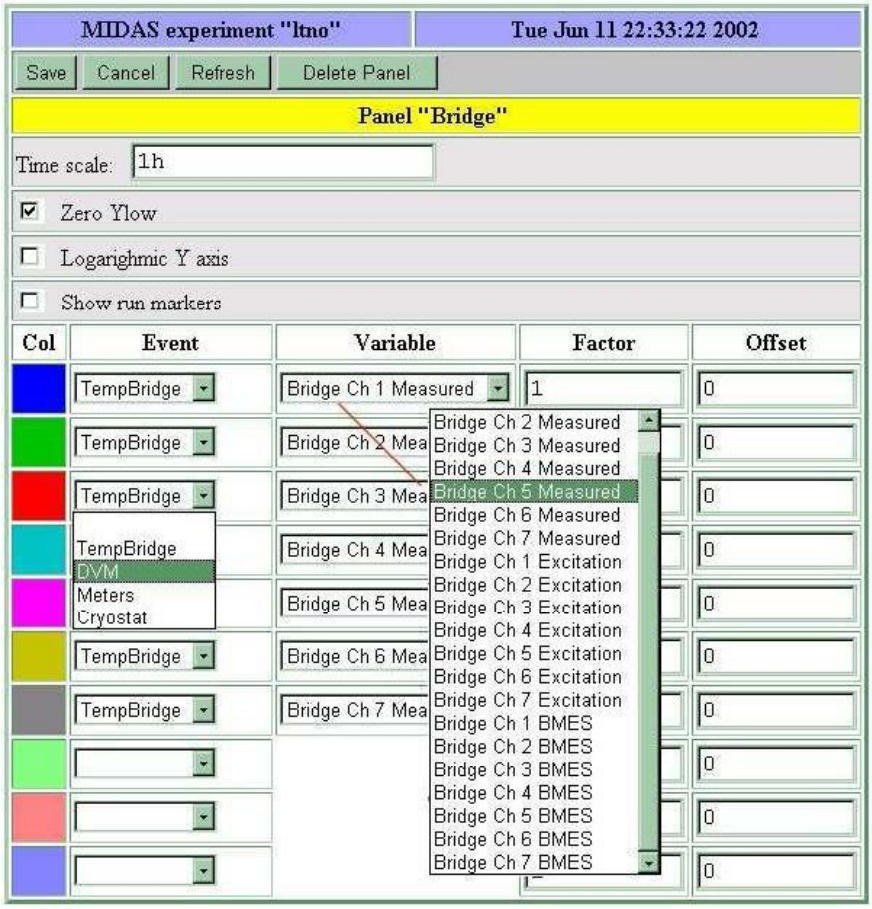


Figure 28: History channel selection Page.

6.12.9 Alarm page

This page reflects the Alarm System settings. It presents the four type of alarms:

- [Evaluated alarms] Triggered by ODB value on given arithmetical condition.
- [Program alarms] Triggered on condition of the state of the defined task.

- [Internal alarms] Trigger on internal (program) alarm setting through the use of the *al_...()* functions.
- [Periodic alarms] Triggered by timeout condition defined in the alarm setting.

6.12.10 Custom page

The Custom page is available since version 1.8.3. It has been improved during version 1.9.5 (mhttpd.c CVS-1.288).

This custom web page provides to the user a mean of creating a secondary personal web page activated within the standard Midas web interface. This custom page can contain specific links to the ODB and therefore present in a more compact way the essential parameter of the controlled experiment. Two mode of operations are available:

- Internal HTML document. : The html code is fully stored in the Online Database (ODB). This page is web editable.
- External referenced HTML document. : ODB contains a link to an external html document.
- Custom Script usage. : External html code with custom script option.

6.12.10.1 Internal HTML document. This page reflects the html content of a given ODB key under the **/Custom/** key. If keys are defined in the ODB under the **/Custom/** the name of the key will appear in the main status page as the **Alias** keys. By clicking on the Custom page name, the content of the **/Custom/**<**page**> is interpreted as html content.

Custom web page with history graph.

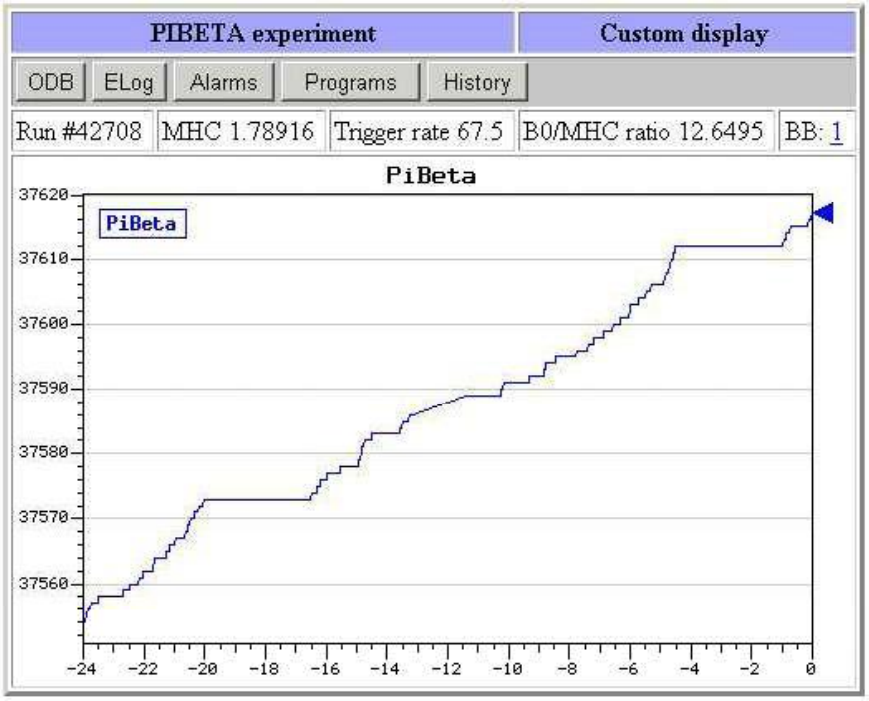


Figure 29: Custom web page with history graph.

The access to the ODB field is then possible using specific HTML tags:

- <**odb src="odb field"**> Display ODB field.
- <**odb src="odb field" edit=1**> Display and Editable ODB field.
- <form method="GET" action="http://hostname.domain:port/CS/<Custom_page_key>"> Define method for key access.
- <meta http-equiv="Refresh" content="60"> Standard page refresh in second.
- <input type=submit name=cmd value=<Midas_page>> Define button for accessing Midas web pages. Valid values are the standard midas buttons (Start, Pause, Resume, Stop, ODB, Elog, Alarms, History, Programs, etc).
- <img src="http://hostname.domain:port/HS/Meterdis.gif&scale=12h&width=300"> Reference to an history page.

ODB /Custom/ html field.

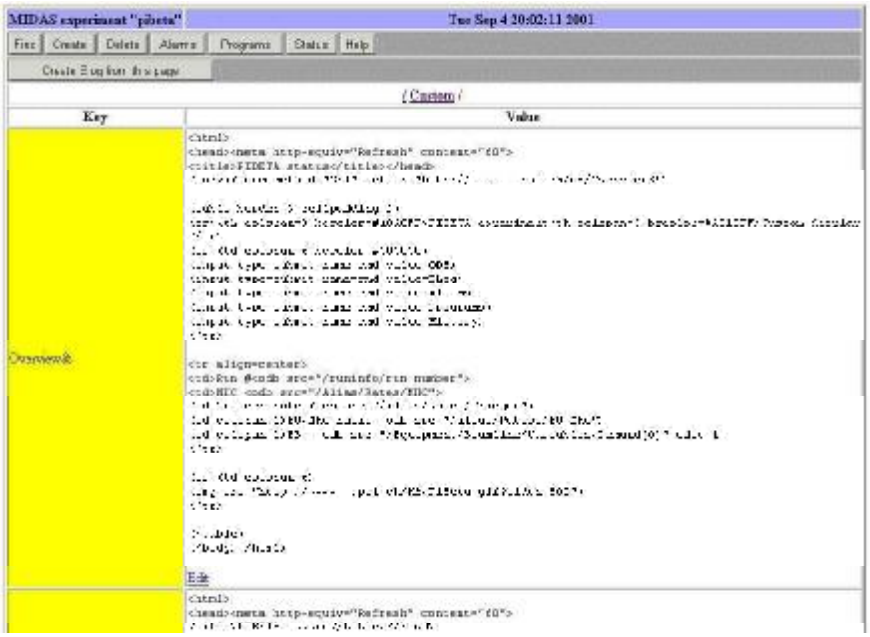


Figure 30: ODB /Custom/ html field.

The insertion of a new Custom page requires the following steps:

- Create an initial html file using your favorite HTML editor.
- Insert the ODB HTML tags at your wish.
- Invoke ODBedit, create the Custom directory, import the html file.
- **Example** of loading the file mcustom.html into odb.

```
Tue> odbedit
[local:midas:Stopped]/>ls
System
Programs
Experiment
Logger
Runinfo
Alarms
Equipment
[local:midas:Stopped]/>mkdir Custom
[local:midas:Stopped]/>cd Custom/
[local:midas:Stopped]/Custom>import mcustom.html
Key name: Test&
[local:midas:Stopped]/Custom>
```

- Once the file is load into ODB, you can **ONLY** edit it through the web (as long as the mhttpd is active). Clicking on the **ODB(button)** ... Custom(Key) ... Edit(Hyperlink at the bottom of the key). The Custom page can also be exported back to a ASCII file using the ODBedit command "export"

```
Tue> odbedit
[local:midas:Stopped]/>cd Custom/
[local:midas:Stopped]/Custom>export test&
File name: mcustom.html
[local:midas:Stopped]/Custom>
```

- The character "&" at the end of the custom key name forces the page to be open within the current frame. If this character is omitted, the page will be spawned into a new frame (default).

- If the custom page name is set to **Status** (no "&") it will become the default midas Web page!

- html code **Example** mcustom.html

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
 <head>
 <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
 <meta name="GENERATOR" content="Mozilla/4.76 [en] (Windows NT 5.0; U) [Netscape]">
 <meta name="Author" content="Pierre-André Amaudruz">
 <title>Set value</title>
 </head>
 <body text="#000000" bgcolor="#FFFFCC" link="#FF0000" vlink="#800080" alink="#0000FF">
 <form method="GET" action="http://host.domain:port/CS/WebLtno&">
 <input type=hidden name=exp value="ltno">
  <center><table CELLSPACING=0 CELLPADDING=0 COLS=3 WIDTH="100%" BGCOLOR="#99FF99" >
  <caption><b><font face="Georgia"><font color="#000099"><font size=+2>LTNO
     Custom Web Page</font></font></font></b></caption>
  <tr BGCOLOR="#FFCC99">
  <td><b><font color="#FF0000">Actions: </font></b>
  <input type=submit name=cmd value=Status>
  <input type=submit name=cmd value=Start>
  <input type=submit name=cmd value=Stop>
  <td>
  <input type=submit name=cmd value=ODB>
  <input type=submit name=cmd value=History>
  <input type=submit name=cmd value=Elog></td>
  <td><div align=right><b>LTNO experiment </b></div>
  </td></tr>
  <tr><td><b>Cryostat section:</b>
  <br>LN2 Bath Level : <odb src="/equipment/cryostat/variables/measured[12]">
  <br>Run# : <odb src="/runinfo/run number" edit=1>
  <br>Run#: <odb src="/runinfo/run number"></td>
  <td WIDTH="100%" BGCOLOR="#009900"><b>RF source section:</b>
  <br>Run#: <odb src="/runinfo/run number"></td>
  <td WIDTH="50%" BGCOLOR="#FF6600"><b>Run section:</b>
  <br>Start Time: <odb src="/runinfo/start time">
```

```
  <br>Stop Time: <odb src="/runinfo/stop time">
  <br>Run#: <odb src="/runinfo/run number"></td>
  </tr>
  <tr>
  <td BGCOLOR="#CC6600"><b>Sucon magnet section:</b>
  <br>Run#: <odb src="/runinfo/run number"></td>

  <td BGCOLOR="#FFCC33"><b>Scalers section:</b>
  <br>Beam Current: <odb src="/equipment/epics/variables/measured[10]">
  <br>Run#: <odb src="/runinfo/run number"></td>

  <td BGCOLOR="#66FFFF"><b>Polarity section:</b>
  <br>Run#: <odb src="/runinfo/run number"></td>
  </tr>
  </table></center>

  <img src="http://host.domain:port/HS/Meterdis.gif?exp=ltno&scale=12h&width=400">
  <img src="http://host.domain:port/HS/Bridge.gif?exp=ltno&scale=12h&width=400">
  <b><i><font color="#000099"><a href="http://host.domain/index.html">
  <br> LTNO help</a></font></i></b>
  </body>
</html>
```

6.12.10.2 External referenced HTML document. The new External referenced HTML document. feature remove the html code size restriction and support multiple custom web page. In addition, to each html document, a dynamic ODB linked image extend the display presentation capability of the controlled experiment.

In the case the custom web page is rather large and complex, it becomes easier to handle such file through normal html editor and skip the reloading of the file in the ODB. (import/export). This is now possible by providing an external reference of the web page in the /Custom directory of the ODB. In addition special ODB settings are available to allow GIF image insertion and ODB fields bars and fillup area superimposed on the image. This powerful new extention brings the mhttpd capability closer to other experiment web control similar to EPICS.

The HTML examples below should operate in conjunction of the standard demo midas example found in midas/examples/experiment. myexpt.html, xcumstom.odb and myexpt.gif can be found in the midas/examples/custom directory.

Using your favorite html editor, you can create a custom page including any of the options described in the Internal HTML document.. Once the mhttpd application is started and connected to a valid Midas experiment, you can activate this page as follow:

```
[local:Default:Stopped]/>pwd
/
[local:Default:Stopped]/>mkdir Custom
[local:Default:Stopped]/>cd Custom
[local:Default:Stopped]/Custom>create string Dewpoint&
```

```
String length [32]: 256
[local:Default:Stopped]/Custom>set Dewpoint& \doc\cooling\dewpoint.html
```

Note: This link refers to a local html document. In the case an external HTML is requires, the definition should be placed under /Alias (see also ODB /Alias Tree).

```
[local:Default:Stopped]/>mkdir Alias
[local:Default:Stopped]/>cd alias
[local:Default:Stopped]/alias>create string WebDewpoint&
String length [32]: 256
[local:Default:Stopped]/alias>set WebDewpoint& "http://www.decatur.de/javascript/dew/index.html"
```

After refreshing the Midas status web page, the link **Dewpoint** should be visible in the top area of the page. The "&" is to prevent a new frame to be displayed (see ODB /Alias Tree). Clicking on it will bring you to your custom html documentation. In the case you want to extend the flexibility of your page by including features such as:

- "live" ODB values position in a particular location of the page.
- "bar level" showing graphically levels or rate etc.
- "color level" where color is used as level indicator. you need to setup specific ODB tree related to a particular page. This overlay of the requested features is done on a GIF file representing you background experimental layout for example. myexpt.html can be found in the examples/custom directory. For the full operation of this custom demo, you'll need to have the frontend "sample frontend" (midas/example/experiment/frontend.c), mlogger, mhttpd running.

Html document **myexpt.html**

```
<html>
 <head>
  <title>MyExperiment Demo Status</title>
  <meta http-equiv="Refresh" content="30">
 </head>
 <body>
   <form name="form1" method="Get" action="/CS/MyExpt&">
   <table border=3 cellpadding=2>
  <tr><th bgcolor="#A0A0FF">Demo Experiment<th bgcolor="#A0A0FF">Custom Monitor/Control</tr>
  <tr><td> <b><font color="#ff0000">Actions: </font></b><input
      value="Status" name="cmd" type="submit"> <input type="submit"
      name="cmd" value="Start"><input type="submit" name="cmd" value="Stop">
  </td><td>
  <center> <a href="http://midas.triumf.ca/doc/html/index.html"> Help </a></center>
  </td></tr>
  <td>Current run #: <b><odb src="/Runinfo/run number"></b></td>
  <td>#events: <b><odb src="/Equipment/Trigger/Statistics/Events sent"></b></td>
  </tr><tr>
  <td>Event Rate [/sec]: <b><odb src="/Equipment/Trigger/Statistics/Events per sec."></b></td>
  <td>Data Rate [kB/s]: <b><odb src="/Equipment/Trigger/Statistics/kBytes per sec."></b></td>
```

```
  </tr><tr>
  <td>Cell Pressure: <b><odb src="/Equipment/NewEpics/Variables/CellPressure"></b></td>
  <td>FaradayCup   : <b><odb src="/Equipment/NewEpics/Variables/ChargeFaradayCup"></b></td>
  </tr><tr>
  <td>Q1 Setpoint: <b><odb src="/Equipment/NewEpics/Variables/EpicsVars[17]" edit=1></b></td>
  <td>Q2 Setpoint: <b><odb src="/Equipment/NewEpics/Variables/EpicsVars[19]" edit=1></b></td>
  </tr><tr>
  <th> <img src="http://localhost:8080/HS/Default/Trigger%20rate.gif?
                exp=default&amp;scale=12h&amp;width=250">
  </th>
  <th> <img src="http://localhost:8080/HS/Default/Scaler%20rate.gif?
                exp=default&amp;scale=10m&amp;width=250"></th>
  </tr>
  <tr><td colspan=2>
  <map name="myexpt.map">
     <area shape=rect coords="140,70, 420,170"
     href="http://midas.triumf.ca/doc/html/index.html" title="Midas Doc">
     <area shape=rect coords="200,200,400,400"
     href="http://localhost:8080" title="Switch pump">
     <area shape=rect coords="230,515,325,600"
     href="http://localhost:8080" title="Logger in color level (using Fill)">
  <img src="myexpt.gif" border=1 usemap="#myexpt.map">
  </map>
  </td></tr>
   </table></form>
  </body>
</html>
```

To activate this HTML document, it has to be defined in the ODB as follow:

```
[local:Default:Stopped]/>cd /Custom
[local:Default:Stopped]/Custom>create string Myexpt&
String length [32]: 256
[local:Default:Stopped]/Custom>set Myexpt& \midas\examples\custom\myexpt.html
```

After refresh, the ODB values will be displayed, the mapping is still not active. as no reference to the gif location has been given yet.

```
[local:Default:Stopped]/Custom>mkdir Images
[local:Default:Stopped]/Custom>cd Images/
[local:Default:Stopped]Images>mkdir myexpt.gif
[local:Default:Stopped]Images>cd myexpt.gif/
[local:Default:Stopped]myexpt.gif>create string Background
String length [32]: 256
[local:Default:Stopped]myexpt.gif>set Background \midas\examples\custom\myexpt.gif
```

After refresh, the file **myexpt.gif** should by visible. The mapping based on myexpt.map is active, hovering the mouse over the boxes will display the associated titles (Midas Doc, Switch pump, etc), By clicking on either box the browser will go to the defined html page specified by the map.

Custom web page with external reference to html document.

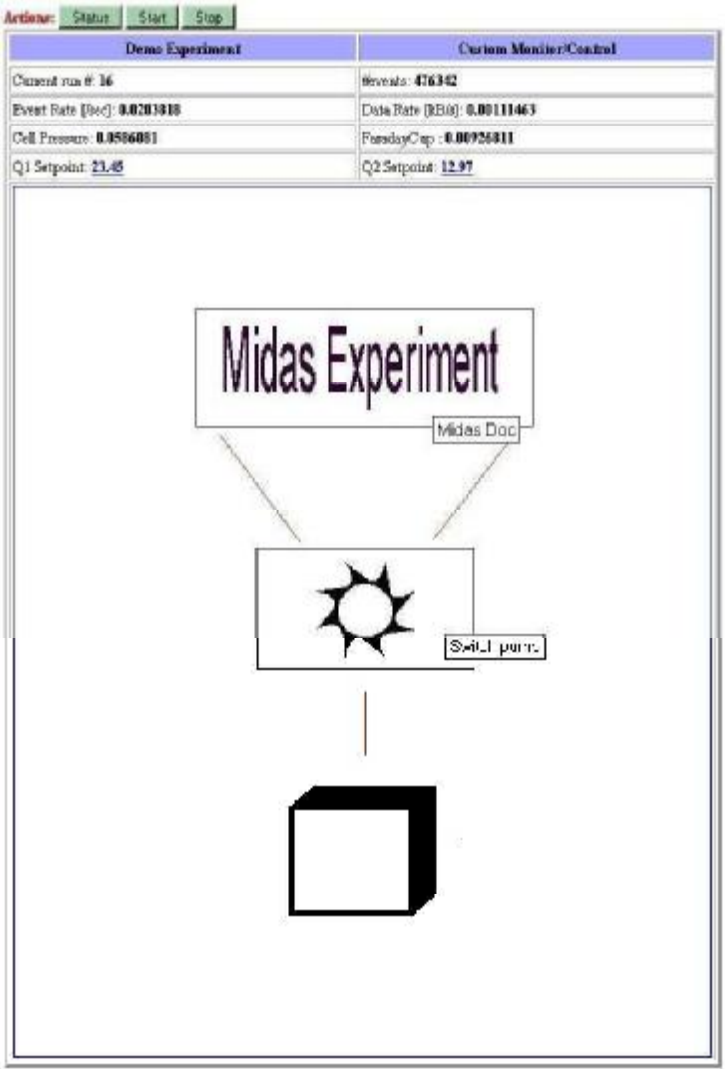


Figure 31: Custom web external to html document.

In addition of these initial features, multiple ODB values can be superimposed at define location on the image. Each entry will have a ODB tree associated to it defining the ODB variable, X/Y position, color, etc...

```
[local:Default:Stopped]myexpt.gif>mkdir Labels
[local:Default:Stopped]myexpt.gif>cd labels
[local:Default:Stopped]Labels>mkdir Rate
```

```
>>>>>>>> Refresh web page <<<<<<<<

12:32:38 [mhttpd] [mhttpd.c:5508:show_custom_gif] Empty Src key for label "Rate"
```

Creating "Labels/<label name>" sub-directory under the gif file name, will automatically at the **next** web page refresh complete its filling with default value for the structure for that label.

```
[local:Default:Stopped]Labels>cd Rate/
[local:Default:Stopped]Rate>ls -l
Key name                        Type    #Val  Size  Last Opn Mode Value
---------------------------------------------------------------------------
Src                             STRING  1     256   2m   0   RWD
Format                          STRING  1     32    2m   0   RWD  %1.1f
Font                            STRING  1     32    2m   0   RWD  Medium
X                               INT     1     4     2m   0   RWD  0
Y                               INT     1     4     2m   0   RWD  0
Align                           INT     1     4     2m   0   RWD  0
FGColor                         STRING  1     8     2m   0   RWD  000000
BGColor                         STRING  1     8     2m   0   RWD  FFFFFF
```

The **Src** should point to a valid ODB Key variable. The X,Y fields position the top left corner of the label. The other fields associated to this label are self-explanatory.

```
[local:Default:Stopped]Rate>set src "/Equipment/Trigger/statistics/kbytes per sec."
[local:Default:Stopped]Rate>set x 330
[local:Default:Stopped]Rate>set y 250
[local:Default:Stopped]Rate>set format "Rate:%1.1f kB/s"
```

Once the initial label is created, the simplest way to extent to multiple labels is to copy the existing label sub-tree and modify the label parameters.

```
[local:Default:Stopped]Labels>cd ..
[local:Default:Stopped]Labels>copy Rate Event
[local:Default:Stopped]Labels>cd Events/
[local:Default:Stopped]Event>set src "/Equipment/Trigger/statistics/events per sec."
[local:Default:Stopped]Event>set Format "Rate:%1.1f evt/s"
[local:Default:Stopped]Event>set y 170
[local:Default:Stopped]Event>set x 250
```

In the same manner, you can create bars used for level representation. This code will setup two ODB values defined by the fields src.

```
[local:Default:Stopped]myexpt.gif>pwd
/Custom/Images/myexpt.gif
[local:Default:Stopped]myexpt.gif>mkdir Bars
[local:Default:Stopped]myexpt.gif>cd bars/
[local:Default:Stopped]Labels>mkdir Rate
```

```
>>>>>>>> Refresh web page <<<<<<<<

14:05:58 [mhttpd] [mhttpd.c:5508:show_custom_gif] Empty Src key for bars "Rate"
[local:Default:Stopped]Labels>cd Rate/
[local:Default:Stopped]Rate>set src "/Equipment/Trigger/statistics/kbytes per sec."
[local:Default:Stopped]Rate>set x 4640
[local:Default:Stopped]Rate>set y 210
[local:Default:Stopped]Rate>set max 1e6
[local:Default:Stopped]Labels>cd ..
[local:Default:Stopped]Labels>copy Rate Events
[local:Default:Stopped]Labels>cd Events/
[local:Default:Stopped]Event>set src "/logger/channles/0/statistics/events written"
[local:Default:Stopped]Event>set direction 1
[local:Default:Stopped]Event>set y 240
[local:Default:Stopped]Event>set x 450
[local:Default:Stopped]Rate>set max 1e6
```

Following the same principle as for the labels, by creating Bars/<bar name>, the structure for the rate will be filled with a default setting after refreshing the custom midas page. The different parameters are self-explanatory.

The last option available is the Fills where an area can be filled with different colors depending on the given ODB value (src parameter). The color selection is mapped by correspondance of the index of the Limit array to the Fillcolor array. Presently the structure is not pre-defined and need to be entered by hand.

```
[/Custom/Images/myexpt.gif/Fills/Level]
Src = STRING : [256] /equipment/Trigger/statistics/events sent
X = INT : 250
Y = INT : 550
Limits = DOUBLE[4] :
[0] 0
[1] 10
[2] 10000
[3] 100000
Fillcolors = STRING[4] :
[8] 00FF00
[8] AAFF00
[8] AA0000
[8] FF0000
```

Custom web page with external reference to html document.

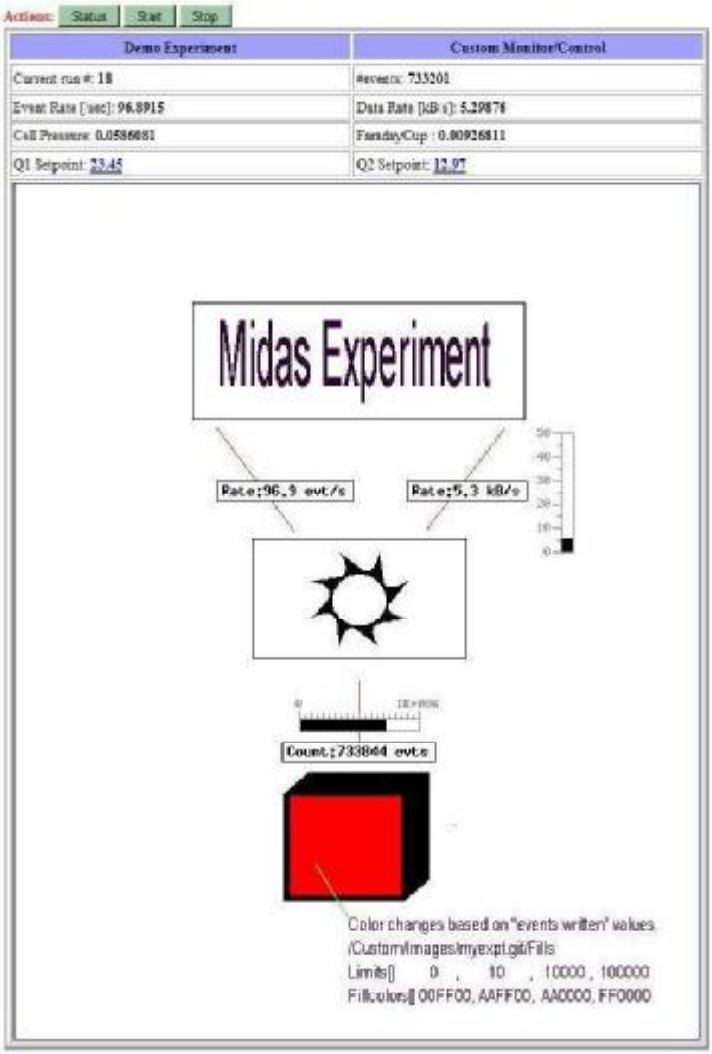


Figure 32: Custom web external to html document.

6.12.10.3 Custom Script usage. From 1.9.5, a new feature has been implemented for the creation of secondary web page activated by an internal or external custom page. This permit to have truly custom page for specific scripts or data display. Use of a new

odb key is required **CustomScript** following the same **Script** syntax.

To be noted that <script> language within the .html source file is possible.

In order to provide a new frame holding input parameters with start script button using the inputs parameters as arguments the setup is the following:

- Create the mybutton.html code as described in the Internal HTML document..
- Create a new custom (internal or External) in ODB under **/Custom/mybutton&**
- Create a new custom script (with argument as described in the myscript.html) in ODB under **/CustomScript/myscript&**

These operations will implement a new button <mybutton> in the main Status Midas web page (same lne as alias). By clicking <**mybutton**> a new frame will be created as described in the *mybutton.html*. By clicking on the <**myscript**> of tht new frame, execution of the script using all the argument above will be performed.

- mybutton.html code

```
<!--Custom web page for runall.
This webpage displays some experiment data,
allows the user to enter some experiment parameters,
and then uses these parameters to run a custom script.-->

<html>
        <head>
        <meta http-equiv="Refresh" content="10">
        <title>RUNALL</title>
        </head>

        <body>
        <form method="GET" action="http://<host>:<port>/CS/mybutton&">
        <input type=hidden name=exp value="default">

        <table border ="3" cellpadding ="5" width ="40%">
        <tr align ="center"<th bgcolor =#A0A0FF>
        MIDAS experiment "default"</th>
        <th bgcolor =#A0A0FF>

 <script>
        var mydate=new Date()
        var year=mydate.getYear()
        if (year < 1000)
          year+=1900
        var day=mydate.getDay()
        var month=mydate.getMonth()
        var daym=mydate.getDate()
        var hour=mydate.getHours()
        var min=mydate.getMinutes()
        var sec=mydate.getSeconds()
        if (daym<10)
          daym="0"+daym
```

```
        if (hour<10)
          hour="0"+hour
        if (min<10)
          min="0"+min
        if (sec<10)
          sec="0"+sec
        var dayarray=new Array("Sun","Mon","Tue","Wed","Thur","Fri","Sat")
        var montharray=new Array("Jan","Feb","Mar","Apr","May","June","July",
        "Aug","Sept","Oct","Nov","Dec")
        document.writeln(dayarray[day], " ", montharray[month], " ", daym," ",
        hour, ":", min, ":", sec," ", year);
        </script>
  </th>
  </tr>

        <tr align ="center"<th colspan ="2" bgcolor=#A0A0A0>
        <input type=submit name=cmd value=Status>
        <input type=submit name=cmd value=ODB></th></tr>
        <tr align ="center"><th colspan = "2" bgcolor=#CCCCFF>
        End of Run Parameters</th></tr>
        <tr align ="center"><th> Key </th>  <th>Value </th></tr>
        <tr align ="center"><td><b>Run number</b></td>
        <td><odb src="/runinfo/run number"></td></tr>
        <tr align ="center"><td><b>Number of Runs</b></td>
        <td><odb src="/customscript/myscript/Number of Runs" edit=1></td></tr>
        <tr align ="center"><td><b>Duration in Hours</b></td>
        <td><odb src="/customscript/myscript/Duration in Hours" edit=1></td></tr>
        <tr align ="center"<th colspan ="2" bgcolor=#D0D0D0>
        <input type=submit name=customscript value="myscript">
        </th></tr>
        </table>
        </form>
        </body>
</html>
```

CustomScript usage.

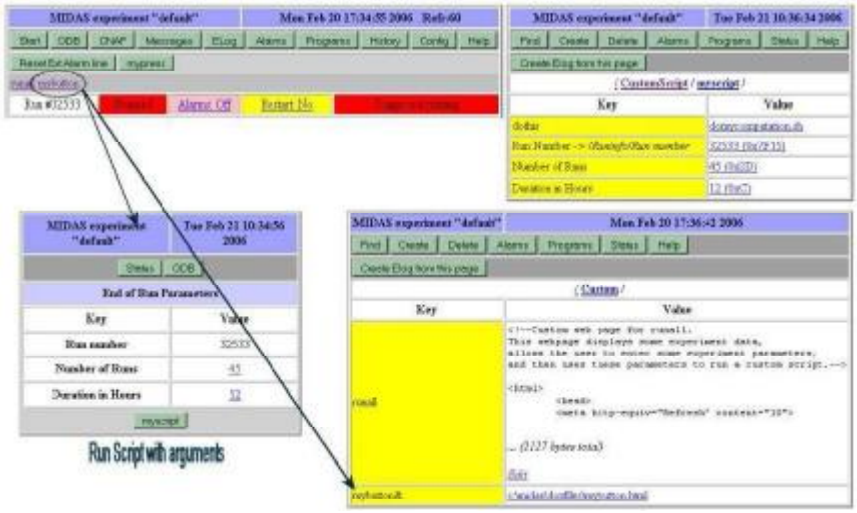


Figure 33: CustomScript usage.

Utilities - Top - Data format

6.13 New Documented Features

Top - Top - Introduction

Some of the midas features are not yet fully documented or even referenced anywhere in the documentation.

This section will maintain an up-to-date information with a log of the latest documentation on past and current features. It will also mention the wish list documentation on current developments.

- Current doc revision: 2.0.0-1
- Software version: 2.0.0

- **[2.0.0]**
  - Update the whole midas package for support of 64 bits machine, OSLFAGS should have -m32 for 32bit (Building Options).
  - Implementation of the external standalone Elog package (External Elog).
  - ODB Buffer size parameter (ODB /Experiment Tree).
  - Fix buffer level handling.

- Improve midas.log, ODB Dump file directory destination (Data_Dir).
- Multiple minor buf fixes
  - Buffer level handling.
  - mdump single bank display.
  - mhttpd (multiple buggies).
- EQ_MULTITHREAD frontend type (see EQ_xxx).
- Ring Buffer fonctions (rb_create(),...) for multi-threading and cascading data transfer.

- **[1.9.6]**
  - Latest tarball : `1.9.5-x` You can retrieve the daily tarball directly from the SVN web interface by clicking on the "tarball" link at the bottom of the main SVN-midas page.
  - Latest RPM : `1.9.2-1`

- **[Before 1.9.6]**
  - Switch from CVS to Subversion for Version control, this change affects Quick Start. Check the new checkout/update commands.
  - New Midas VME standard functions implementation in **mvmestd.h**.
  - **MIDASSYS** environment variable now required for building /examples/experiment and /examples/hbookexpt
  - New **/drivers** tree structure includes **camac** , **vme** , **fastbus**.
  - New make option for minimal installation. This permits root installation of mserver, mcleanup, dio and mhttpd only.

    ```
    > make minimal_install
    ```

- **[1.9.5-2]**
  - XML ODB format
  - Separate xml SVN path for building Midas required. This package can be extracted the same way as Midas. It has to reside at the same level as Midas.

- **[1.9.5-1]**
  - Custom page improvement. Implementation of external file.html and dynamic linked graphic to ODB values.

- **[1.9.5]**

- When upgrading to 1.9.5 , *ALL* midas applications including user applications needs to be rebuild *AND* the **ODB.SHM** (.ODB.SHM) shared memory need to be removed. Prior the removal of the ODB.SHM, the ODB database can be saved in ASCII format for later restoration.
- Run Transition Sequence changed to multiple level scheme.
- odbedit_task support of XML format for ODB dump.
- Large File support (>2GB) from mlogger task application.
- Folder Root Histogram support within mana.
- mevb task application.
- New Midas Frontend application argument for Event Builder option (-i index).
  - Documentation on "Tests" results from analyzer.
- mySQL support from mlogger task.
- Increase system wide parameters values (see midas.h).
- Fix numerous small annoying bugs...
- Improve debugging messages in mserver -d (/tmp/mserver.log).

- **[<1.9.5]**
  - In writing
    - Epics Slow Control documentation

- Introduce MIDASSYS environment variable
- Analyzer documention revision MIDAS Analyzer
- Watchdog bug fix (RH9.0)

- **Restructured Midas distribution**
  - In the same effort as the documentation, the midas tree and CVS have been modified. The `download area` now contains separate directories for doc, add-ons, publications etc.

[DOCUMENTATION in progress]

- A large effort has been put on the documentation for switching from the **DOC++** to `Doxygen` We feel the cross-referencing to the source code is excellent and hopefully will server better its purpose. Currently the MIDAS Analyzer is not complete as well as the Quick Start. This **Doxygen** related files will be made accessible for better update.

- **[Midas Short Course]**
  - During the RealTime Conference 2003 held in Montreal, a short course was offered to introduce the Midas DAQ to the audience. This course (.ppt, .pdf) is now part of the Midas distribution and can be found under the doc/course/ directory as 2 files (part1, part2). The Part 1 describes the basic of the system and its implementation, while part 2 lists specific features. `Part1.pdf, Part2.pdf` .

- **[1.9.3]**
  - Support for ROOT files.
  - mlogger task : New Data format **ROOT** and corresponding file extension **root**
  - rmidas task : Initial Root/Midas GUI for Histogram and Run control.
  - MIDAS Analyzer : New framework for Online/Offline Root analysis using socket connection.
  - **Makefile** for ROOT, remove **MANA_LITE**, create HAVE_ROOT, HAVE_HBOOK.
  - New Analyzer **mana**, **hmana**, **rmana** depending on the type of package.

- **[1.9.2]**
  - odbedit: <tab> completion is working with flags too, "Load" protect the data dir if changed.
  - lazylogger task : This task has been improved for tape manilupation as well as messages display. It has also now extra fiilds for shell scripts when the tape rewinds. It supports also split run capability when running multiple instance of the task. Please refer to the documentation for explanation of the new fields.
  - mlxspeaker: Added possible system call to wav file for "beeping" user before message.
  - mhist: Add index range for -i with -v.
  - eventbuilder: Revised version with user code scheme. Still in a development stage.
  - cm_cleanup() if you were using this call, you need now to provide an empty char arg to make it compatible.

- **[1.9.1]**
  - This version addresses several bugs reported in the web interface, history, logger, odbedit and implements new features in particular for the history pages on web interface. The detail list of the modifications can be found in `CHANGELOG` .

    - **[EQ_FRAGMENTED]** Possibility to send extremely large event through the system without modification of the system configuration (see The Equipment structure)
    - **[logger subdir option]** Allows to redirect the data files to a subdirectory based on the time of the creation of the data file (see ODB /Logger Tree).
    - Option for building an analyzer without the CERN library (HBOOK) (see Midas build options and operation considerations).
    - **[MOD. REQ.]** This release requires several modifications in the user code in order to compile the 1.9.1.
        1. **[db_get_value() function]** Requires an extra parameter see Midas Code and Libraries.
        2. **[max_event_size_frag]** Required in all the frontend code as follow:

```
// maximum event size produced by this frontend
INT max_event_size = 10000;
// maximum event size for fragmented events (EQ_FRAGMENTED)
INT max_event_size_frag = 5*1024*1024;
```

  - **[/Logger tree]** As this tree includes new field, you will need to recreate this tree.
  - **[general]** It is wise to create a fresh ODB when switching to 1.9.1 version. This can be done by:
      1. removing all attached midas client to your experiment
      2. saving the current ODB to a file
      3. removing all shared memory files (hidden files .*.SHM)
      4. creating new ODB (odbedit -s size)
      5. trimming the odb save file to keep user specific structures (if any).
      6. restoring the trimmed odb file.

- **[<1.9.1]**
  - Hopefully nobody is still running an older version.

Top - Top - Introduction

6.14 ODB Structure

Internal features - Top - Data format

The Online Database contains information that system and user wants to share. Basically all transactions for experiment setup and monitoring go through the ODB. It also contains some specific system information related to the "Midas client" currently involved in an experiment (/system).

Each ODB field or so called **KEY** is accessible by the user through either an interactive way (see odbedit task) or by C-programming (see functions db_xxx in Midas Code and Libraries).

The ODB information is stored in a "tree/branch" structure where each branch refers to a specific set of data. On the first invocation of the database (first Midas application) a minimal system record will be created. Later on each application will add its own set of parameters to the database depending on its requirement. For instance, starting the ODB for the first time, the tree **/Runfinfo**, /Experiment, /System will be created. The application mlogger task will add its own tree **/Logger/**...

As mentioned earlier, ODB is the main communication platform between any Midas application. As such, the content of the ODB is application dependent. Several "dormant" trees can be awaken by the user in order to provide extra flexibility of the system. Such "dormant" tree are **Alias**, Script, Edit on Start , Security, Run parameters .

- ODB /System Tree
- ODB /RunInfo Tree
- ODB /Equipment Tree
- ODB /Logger Tree
- ODB /Experiment Tree
- ODB /History Tree
- ODB /Alarms Tree
- ODB /Script Tree
- ODB /Alias Tree
- ODB /Elog Tree
- ODB /Programs Tree
- ODB /Lazy Tree
- ODB /EBuilder Tree
- ODB /Custom Tree

6.14.1 ODB /System Tree

The system tree contains information specific to each "Midas client" currenltly connected to the experiment. This information is not primarly for the user but may be informative in some respect to the reader.

```
[host:expt:Stopped]/>ls -r -l /system
Key name                        Type    #Val  Size  Last Opn Mode Value
------------------------------------------------------------------------
System                          DIR
    Clients                     DIR
        29580                   DIR
            Name                STRING  1     32    17h  0   R    decay
            Host                STRING  1     256   17h  0   R    host1
            Hardware type       INT     1     4     17h  0   R    42
            Server Port         INT     1     4     17h  0   R    1227
            Transition Mask     DWORD   1     4     17h  0   R    329
            Deferred Transition DWORD   1     4     17h  0   R    6
            RPC                 DIR
                16000           BOOL    1     4     17h  0   R    y
                16001           BOOL    1     4     17h  0   R    y
        29638                   DIR
            Name                STRING  1     32    17h  0   R    MStatus
            Host                STRING  1     256   17h  0   R    host1
            Hardware type       INT     1     4     17h  0   R    42
            Server Port         INT     1     4     17h  0   R    1228
            Transition Mask     DWORD   1     4     17h  0   R    0
            Deferred Transition DWORD   1     4     17h  0   R    0
        29810                   DIR
            Name                STRING  1     32    17h  0   R    Nova_029810
            Host                STRING  1     256   17h  0   R    host
            Hardware type       INT     1     4     17h  0   R    42
            Server Port         INT     1     4     17h  0   R    1235
            Transition Mask     DWORD   1     4     17h  0   R    0
        29919                   DIR
            Name                STRING  1     32    17h  0   R    Epics
            Host                STRING  1     256   17h  0   R    host
            Hardware type       INT     1     4     17h  0   R    42
            Server Port         INT     1     4     17h  0   R    1237
            Transition Mask     DWORD   1     4     17h  0   R    329
            Deferred Transition DWORD   1     4     17h  0   R    0
            RPC                 DIR
                16000           BOOL    1     4     17h  0   R    y
                16001           BOOL    1     4     17h  0   R    y
        12164                   DIR
            Name                STRING  1     32    6s   0   R    ODBEdit
            Host                STRING  1     256   6s   0   R    host2
            Hardware type       INT     1     4     6s   0   R    42
            Server Port         INT     1     4     6s   0   R    4893
            Transition Mask     DWORD   1     4     6s   0   R    0
            Deferred Transition DWORD   1     4     6s   0   R    0
            Link timeout        INT     1     4     6s   0   R    10000
    Client Notify               INT     1     4     6s   0   RWD  0
    Prompt                      STRING  1     256   >99d 0   RWD  [%h:%e:%S]%p>
    Tmp                         DIR
```

- [Remark 1] The key **Prompt** sets up the prompt of the ODBEdit program.

```
odbedit
[local:midas:Stopped]/>cd /System/
[local:midas:Stopped]/System>ls
Clients
Tmp
Client Notify                   0
Prompt                          [%h:%e:%S]%p>

[local:midas:Stopped]/System>set Prompt my_prompt>
my_prompt>set Prompt [Host:%h-Expt:%e:State:%s]Path:%p>
[Host:local-Expt:midas-State:S]Path:/System>set Prompt [Host:%h-Expt:%e-State:%S]Path:%p>
[Host:local-Expt:midas-State:Stopped]Path:/System>
```

6.14.2 ODB /RunInfo Tree

This branch contains system information related to the run information. Several time fields are available for run time statistics.

```
odb -e expt -h host
[host:expt:Running]/>ls -r -l /runinfo
Key name                        Type    #Val  Size  Last Opn Mode Value
------------------------------------------------------------------------
Runinfo                          DIR
    State                        INT     1     4     2h   0   RWD  3
    Online Mode                  INT     1     4     2h   0   RWD  1
    Run number                   INT     1     4     2h   0   RWD  8521
    Transition in progress       INT     1     4     2h   0   RWD  0
    Requested transition         INT     1     4     2h   0   RWD  0
    Start time                   STRING  1     32    2h   0   RWD  Thu Mar 23 10:03:44 2000
    Start time binary            DWORD   1     4     2h   0   RWD  953834624
    Stop time                    STRING  1     32    2h   0   RWD  Thu Mar 23 10:03:33 2000
    Stop time binary             DWORD   1     4     2h   0   RWD  0
```

- **[State]** Specifies in which state the current run is. The possible states are 1: STOPPED, 2: RUNNING, 3: PAUSED.
- **[Online Mode]** Specifies the expected acquisition mode. This parameter allows the user to detect if the data are coming from a "real-time" hardware source or from a data save-set. Note that for analysis replay using "analyzer" this flag will be switched off.
- **[Run number]** Specifies the current run number. This number is automatically incremented by a successful run start procedure.
- **[Transition in progress]** Specifies the current internal state of the system. This parameter is used for multiple source of "run start" synchronization.

- **[Requested transition]** Specifies the current internal of the Deferred Transition state of the system.
- **[Start Time]** Specifies in an ASCII format the time at which the last run has been started.
- **[Start Time binary]** Specifies in a binary format at the time at which the last run has been started This field is useful for time interval computation.
- **[Stop Time]** Specifies in an ASCII format the time at which the last run has been stopped.
- **[Stop Time binary]** Specifies in a binary format the time at which the last run has been stopped. This field is useful for time interval computation.

6.14.3 ODB /Equipment Tree

Every frontend create a entry under the /Equipment tree. The name of the sub-tree is taken from the frontend source code in the equipment declaration (frontend.c). More detailed explanation of the composition of that tree will be found throughout this document.

```
{
 "DspecCheck",      // equipment name
 ...
 ,
{
 "Scaler",      // equipment name
 ...
 ,
```

Example:

```
Key name                        Type    #Val  Size  Last Opn Mode Value
------------------------------------------------------------------------------
HistoCheck                      DIR
DSpecCheck                      DIR
HistoPoll                       DIR
HistoEOR                        DIR
DSpecEOR                        DIR
Scaler                          DIR
SuconMagnet                     DIR
TempBridge                      DIR
Cryostat                        DIR
Meters                          DIR
RFSource                        DIR
DSPec                           DIR
```

The equipment tree is then split in several sections which by default the system creates.

- Common : Contains the system information. Should not be overwritten by the user.
- Variables : Contains the equipment data if enabled (see below).
- Settings : Contains the equipment specific information that the user may want to maintain. In the case of a Slow Control System equipment, extended tree structure is created by the system.
- Statistics : Contains equipment statistics information such as event taken, event rate, data rate.

```
[local:S]ls -l -r /equipment/scaler
Key name                        Type    #Val  Size  Last Opn Mode Value
--------------------------------------------------------------------------------
Scaler                          DIR
    Common                      DIR
        Event ID                WORD    1     2     16h  0   RWD  1
        Trigger mask            WORD    1     2     16h  0   RWD  256
        Buffer                  STRING  1     32    16h  0   RWD  SYSTEM
        Type                    INT     1     4     16h  0   RWD  1
        Source                  INT     1     4     16h  0   RWD  0
        Format                  STRING  1     8     16h  0   RWD  MIDAS
        Enabled                 BOOL    1     4     16h  0   RWD  y
        Read on                 INT     1     4     16h  0   RWD  377
        Period                  INT     1     4     16h  0   RWD  1000
        Event limit             DOUBLE  1     8     16h  0   RWD  0
        Num subevents           DWORD   1     4     16h  0   RWD  0
        Log history             INT     1     4     16h  0   RWD  0
        Frontend host           STRING  1     32    16h  0   RWD  midtis03
        Frontend name           STRING  1     32    16h  0   RWD  feLTNO
        Frontend file name      STRING  1     256   16h  0   RWD  C:\online\sc_ltno.c
    Variables                   DIR
        SCLR                    DWORD   6     4     1s   0   RWD
                                        [0]             0
                                        [1]             0
                                        [2]             0
                                        [3]             0
                                        [4]             0
                                        [5]             0
        RATE                    FLOAT   6     4     1s   0   RWD
                                        [0]             0
                                        [1]             0
                                        [2]             0
                                        [3]             0
                                        [4]             0
                                        [5]             0
    Statistics                  DIR
        Events sent             DOUBLE  1     8     1s   0   RWDE 370
        Events per sec.         DOUBLE  1     8     1s   0   RWDE 0.789578
        kBytes per sec.         DOUBLE  1     8     1s   0   RWDE 0.0678543
```

6.14.4 ODB /Logger Tree

The /Logger ODB tree contains all the relevant information for the Midas logger utility (mlogger task) to run properly. This utility provides the mean of storing the physical data retrieved by the frontend to a storage media. The user has no code to write in order for the system to operate correctly. Its general behavior can be customized and multiple logging channels can be defined. The application supports so far three type of storage devices i.e.: *Disk*, *Tape* and *FTP* channel.

Default settings are created automatically when the logger starts the first time:

```
Key name                        Type    #Val  Size  Last Opn Mode Value
------------------------------------------------------------------------
Logger                          DIR
    Data dir                    STRING  1     256   4h   0   RWD  /scr0/spring2000
    Message file                STRING  1     256   22h  0   RWD  midas.log
    Write data                  BOOL    1     4     2h   0   RWD  n
    ODB Dump                    BOOL    1     4     22h  0   RWD  y
    ODB Dump File               STRING  1     256   22h  0   RWD  run%05d.odb
    Auto restart                BOOL    1     4     22h  0   RWD  y
    Tape message                BOOL    1     4     15h  0   RWD  y
    Channels                    DIR
        0                       DIR
            Settings            DIR
                Active          BOOL    1     4     1h   0   RWD  y
                Type            STRING  1     8     1h   0   RWD  Disk
                Filename        STRING  1     256   1h   0   RWD  run%05d.ybs
                Format          STRING  1     8     1h   0   RWD  YBOS
                ODB Dump        BOOL    1     4     1h   0   RWD  y
                Log messages    DWORD   1     4     1h   0   RWD  0
                Buffer          STRING  1     32    1h   0   RWD  SYSTEM
                Event ID        INT     1     4     1h   0   RWD  -1
                Trigger Mask    INT     1     4     1h   0   RWD  -1
                Event limit     DWORD   1     4     1h   0   RWD  0
                Byte limit      DOUBLE  1     8     1h   0   RWD  0
                Tape capacity   DOUBLE  1     8     1h   0   RWD  0
    Subdir format   STRING  1     32    7h   0   RWD  %Y%m%d
    Current filenameSTRING  1     256   7h   0   RWD  20020605\run00078.mid
      Statistics          DIR
                Events written  DOUBLE  1     8     1h   0   RWD  0
                Bytes written   DOUBLE  1     8     1h   0   RWD  0
                Bytes written toDOUBLE  1     8     1h   0   RWD  3.24316e+11
                Files written   INT     1     4     1h   0   RWD  334
```

From Midas version 1.9.5, the logger has the possibility to store information to a mySQL database. This option is an alternative to the **runlog.txt** update hanled by the analyzer. The two main advantages using the SQL are:

- The recording is done by the logger and therfore independent of the user analyzer.
- The definition of the parameters to be recorded in the database is entirely setup in the ODB under /logger/SQL. This SQL option is enabled by defining at build

time the preprocessor flag HAVE_MYSQL. This option when enabled will create a sub tree *SQL* under /Logger in the ODB. This tree contains information for mySQL access with predefined mySQL database name *Midas* and table *Runlog*. Under 2 dedicated sub directories i.e: Link_BOR and Link_EOR, predefined links exists which will be used respectively at BOR and EOR for storing into the database. These elements are ODB links allowing the user to extend the list with any parameter of the ODB database. This logger mySQL option is to replace or complement the *runlog.txt* functionality of the ana_end_of_run() function from the analyzer.c.

```
[local:midas:S]/Logger>ls -lr SQL
Key name                        Type    #Val  Size  Last Opn Mode Value
--------------------------------------------------------------------------
SQL                             DIR
    Create database             BOOL    1     4     27s  0   RWD  n
    Write data                  BOOL    1     4     27s  0   RWD  n
    Hostname                    STRING  1     80    27s  0   RWD  localhost
    Username                    STRING  1     80    27s  0   RWD  root
    Password                    STRING  1     80    27s  0   RWD
    Database                    STRING  1     32    27s  0   RWD  midas
    Table                       STRING  1     80    27s  0   RWD  Runlog
    Links BOR                   DIR
        Run number              LINK    1     20    58s  0   RWD  /Runinfo/Run number
        Start time              LINK    1     20    58s  0   RWD  /Runinfo/Start time
    Links EOR                   DIR
        Stop time               LINK    1     19    4m   0   RWD  /Runinfo/Stop time
```

- [Data dir] Specifies in which directory files produced by the logger should be written. Once the Logger in running, this Data_Dir will be pointing to the location of the midas.log , ODB dump files, history files, message files. In the case of multiple logging channels, the data path for all the channels is defaulted to the same location. In the case where specific directory has to be assigned to each individual logging channel, the field **/logger/channel/<x>/Settings/Filename** can contain the full path of the location of the .mid, .ybs, .asc file. By finding the OS specific **SEPARATOR_DIR** ("/", "\"). The field **Filename** will overwite the global Data_Dir setting for that particular channel.
  - [History Dir] This field is optional and doesn't appear by default in the logger. If present the location of the History system files is reassigned to the defined path instead of the default Data_Dir .
  - [Elog Dir] This field is optional and doesn't appear by default in the logger. If present the location of the Electronic Logbook files is reassigned to the defined path instead of the default Data_Dir.
  - [Message file] Specifies the file name for the log file which contains all messages from the MIDAS message system. The message log file is a simple ASCII file, which can be viewed at any time to see a history of what happened in an experiment.

  - **2.0.0** The location of the ODB dump files can now be specified in this field. If the string contains a DIRECTORY_SEPARATOR, is it considered as an absolute path.

– [Write data] Global flag which turns data logging on and off for all channels. It can be set to zero temporarily to make a short test run without data logging. The key "Write data?" is predefined logger key for enabling data logging. This action can be overridden by setting the active key to 1.

– [ODB Dump] Specifies if a dump of the complete ODB should be written to the file specified by ODB Dump File.

– [ODB Dump File] At the end of each run. If the file name contains a "%", this gets replaced by the current run number similar to the printf() C function. The format specifier 05d from above would be evaluated to a five digit run number with leading zeros like run00002.odb. The ODB dump file is in ASCII format and can be used for off-line analysis to check run parameters etc. For a description of the ASCII format see db_copy().

  - **2.0.0** The location of the ODB dump files can now be specified in this field. If the string contains a DIRECTORY_SEPARATOR, is it considered as an absolute path

```
[local:Default:S]/Logger>ls
Data dir                        \online\
Message file                    midas.log
Auto restart                     n
Write data                      y
ODB Dump                        n
ODB Dump File                   run%05d.odb
Tape message                    y
Channels
[local:Default:S]/Logger>set OD
ODB Dump
ODB Dump File
[local:Default:S]/Logger>set "ODB Dump File" "/mypath/run%06d.odb"
[local:Default:S]/Logger>ls
Data dir                        \online\
Message file                    midas.log
Auto restart                    n
Write data                      y
ODB Dump                        n
ODB Dump File                   /mypath/run%06d.odb
Tape message                    y
Channels
```

– [Auto restart] When this flag is one, a new run gets automatically restarted when the previous run has been stopped by the logger due to an event or byte limit.

– [Tape message] Specifies if tape messages during mounting and writing of EOF marks are generated. This can be useful for slow tapes to inform all users in a counting house about the tape status.

– [channels] Sub-directory which contains settings for individual channels. By default, only channel "0" is created. To define other channels, an existing channel can be copied:

```
[loca]]Logger>cd channels
[local]Channels>ls
0
[local]Channels>copy 0 1
[local]Channels>ls
0
1
```

The Settings part of the channel tree has the following meaning:

- [active] turns a channel on (1) or off (0). Data is only logged to channels that are active.
- [Type] Specify the type of media on which the logging should take place. It can be Disk, Tape or FTP to write directly to a remote computer via FTP.
- [Filename] Specify the name of a file in case of a disk logging, where 05d is replaced by the current run number the same way as for the ODB dump files. In the case of a tape logging, the filename specifies a tape device like /dev/nrmt0 or /dev/nst0 under UNIX or \\.\tape0 under Windows NT.
  - In FTP mode, the filename specifies the access information for the FTP server. It has the following format:

    ```
    <host name>, <port number>, <user name>, <password>, <directory>, <file name>
    ```

    The normal FTP port number is 21 and 1021 for a Unitree Archive like the one used at the Paul Scherrer Institute. By using the FTP mode, a back-end computer can directly write to the archive.

    ```
    myhost.my.domain,21,john,password,/usr/users/data,run%05d.mid
    ```
- [Format] Specifies the format to be used for writing the data to the logging channel. It can one of the five value: MIDAS, YBOS, ROOT, ASCII and DUMP. The MIDAS and YBOS binary formats Midas format and YBOS format, respectively. The extention for the file name has to match one of the following.
  - .mid for **MIDAS**
  - .ybs for **YBOS**
  - .root for **ROOT**
  - .asc for **ASCII**
  - .txt for **DUMP**

- The ASCII format converts events into readable text format which can be easily analyzed by programs which have problems reading binary data. While the ASCII format tries to minimize the file size by printing one event per line, the DUMP format gives a very detailed ASCII representation of the event including bank information, serial numbers etc, it should be used for diagnostics. Consistency of this type of format has to be maintained between the frontend declaration and the logger.

- [ODB Dump] Specifies the complete dump of the ODB to the logging channel before and after every run. The ODB content is dumped in one long ASCII

string reflecting the status at begin-of-run event and at end-of-run event. These special events have an ID of EVENT_ID_BOR and EVENTID_EOR and a serial number equals to the current run number. An analyzer in the off-line analysis stage can restore the ODB to its online state.

- [Log messages] This is a bit-field for logging system messages. If a bit in this field is set, the according system message is written to the logging channel as a message event with an ID of EVENT_ID_MESSAGE (0x8002). The bits are 1 for error, 2 for info, 4 for debug, 8 for user, 16 for log, 32 for talk, 64 for call messages and 255 to log all messages. For an explanation of these messages refer to Buffer Manager, Event ID and Trigger .

- [Mask] Specify which events to log. See Frontend code to learn how events are selected by their ID and trigger mask. To receive all events, -1 is used for the event ID and the trigger mask. By using a buffer other than the "SYSTEM" buffer, event filters can be realized. An analyzer can request all events from the "SYSTEM" buffer, but only write acceptable events to a new buffer called "FILTERED". When the logger request now only events from the new buffer instead of the "SYSTEM" buffer, only filtered events get logged.

- [Event limit, Byte limit and Tape capacity] These fields can be used to stop a run when set to a non-zero value. The statistics values Events written, Bytes written and Bytes written total are checked respectively against these limits. When one of these condition is reached, the run is stopped automatically by the logger. Updates of the statistics branch is performed automatically every so often. This branch contains the number of events and bytes written. These two keys are cleared at the beginning of each run. The **Bytes written total** and **Files written** keys are only reset when a tape is rewound with the ODBEdit command rewind. The Bytes written total entry can therefore be used as an indicator if a tape is full. The Files written entry can be used off-line to determine how many files on tape have to be skipped in order to reach a specific run.

- [Subdir format, Current filename] In the case the **Subdir format** is not empty, this field will enable the placement of the data log file into a sub directory. The name of this subdirectory is composed by the given **Subdir** format string. Its format follows the definition of the system call strftime() . Ordinary characters placed in the format string are copied to s without conversion. Conversion specifiers are introduced by a '%' character, and are replaced in s as follows for the most used one:

  - Y : Year (ex: 2002)
  - y : Year (range:00..99)
  - m : Month (range: 01..12)
  - d : Day (range: 00..31) The other characters are: a, A, b, B, c, C, d, D, e, E, G, g, h, H, I, j, k, l, m, M, n, O, p, P, r, R, s, S, t, T, u, U, V, w, W, x, X, y, Y, z, Z, +, %. (See man strftime() for explanations).

- [Current filename] will reflect the full path of the saved data file.

6.14.5 ODB /Experiment Tree

Under this tree, the Midas system stores special features for the user in order to facilitate his job on controlling a run. Initially only one empty key is defined labeled **Name** for the experiment name. The user can create four system keys in order to provide extra run control flexibility i.e.: **"Run Parameter/"**, **"Edit on Start/"**, **"Lock when running/"** and **"Security/"**.

- **2.0.0**, this directory can specify the event buffer size for each buffer involved in the experiment. By default the event buffer is named *SYSTEM*. Its default size is 2MB. This new parameter may be required to optimize the memory usage at the frontend level in case large data transfer is needed. This method work for all MIDAS buffers, except for ODB, where the size has to be specified at creation time using the odbedit command "-s" argument. There is no need to increase the SYSMSG.SHM buffer as it is used only for messages.

  1. Shutdown all MIDAS programs, delete the old .SYSTEM.SHM files sitting in the directory specified by either the exptab or $MIDAS_DIR, use ipcrm for share memory segment removal.
  2. Run odbedit, go to experiment, create a directory key "Buffer Sizes", create a DWORD key of the buffer name to be increased.

     ```
     C:\online>odbedit
     [local:Default:S]/>cd Experiment/
     [local:Default:S]/Experiment>mkdir "Buffer Sizes"
     [local:Default:S]/Experiment>cd "Buffer Sizes/
     [local:Default:S]Buffer Sizes>create DWORD SYSTEM
     [local:Default:S]Buffer Sizes>set SYSTEM 4000000
     ```

  3. Starts the rest of the MIDAS programs. Check that the buffer has the correct size by looking at the size of .SYSTEM.SHM (unix, ipcs), SYSTEM.SHM (windows).

     ```
     Key name                        Type    #Val  Size  Last Opn Mode Value
     ------------------------------------------------------------------------
     Experiment                      DIR
         Name                        STRING  1     32    22s  0   RWD  chaos
         Run Parameter               DIR
             Beam Polarity           STRING  1     256   2h   0   R    negative
             Beam Momentum           FLOAT   1     4     2h   0   R    91
             2LT: log file name?     STRING  1     256   2h   0   R    cni05
             1LT: file name?         STRING  1     256   2h   0   R    files.cni.zero
             Comment                 STRING  1     256   2h   0   R    ch2 target
             Target Angle            FLOAT   1     4     2h   0   R    0
     ```

```
    Target Material         STRING  1     256   2h   0   R    ch2
Edit on start               DIR
    Beam Momentum           FLOAT   1     4     2h   0   R    91
    Beam Polarity           STRING  1     256   2h   0   R    negative
    Target Material         STRING  1     256   2h   0   R    ch2
    Target Angle            FLOAT   1     4     2h   0   R    0
    1LT: file name?         STRING  1     256   2h   0   R    files.cni.zero
    Trigger 2               BOOL    1     4     2h   0   RWD  n
    2LT: log file name?     STRING  1     256   2h   0   R    cni05
    Comment                 STRING  1     256   2h   0   R    ch2 target
    Write data              BOOL    1     4     2h   0   RWD  y
Lock when running           DIR
    Run Parameter           DIR
        Beam Polarity       STRING  1     256   2h   0   R    negative
        Beam Momentum       FLOAT   1     4     2h   0   R    91
        2LT: log file name? STRING  1     256   2h   0   R    cni05
        1LT: file name?     STRING  1     256   2h   0   R    files.cni.zero
        Comment             STRING  1     256   2h   0   R    ch2 target
        Target Angle        FLOAT   1     4     2h   0   R    0
        Target Material     STRING  1     256   2h   0   R    ch2
Security                    DIR
    Password                STRING  1     32    16h  0   RWD  #@D&%F56
    Allowed hosts           DIR
        host.sample.domain  INT     1     4     >99d 0   RWD  0
        pierre.triumf.ca    INT     1     4     >99d 0   RWD  0
        pcch02.triumf.ca    INT     1     4     >99d 0   RWD  0
        koslx1.triumf.ca    INT     1     4     >99d 0   RWD  0
        koslx2.triumf.ca    INT     1     4     >99d 0   RWD  0
        vwchaos.triumf.ca   INT     1     4     >99d 0   RWD  0
        koslx0.triumf.ca    INT     1     4     >99d 0   RWD  0
    Allowed programs        DIR
        mstat               INT     1     4     >99d 0   RWD  0
        mhttpd              INT     1     4     >99d 0   RWD  0
    Web Password            STRING  1     32    16h  0   RWD  pon4@#@%SSDF2
Name                        STRING  1     32    4m   0   RWD  Default
Buffer Sizes                DIR
    SYSTEM                  DWORD   1     4     4m   0   RWD  4000000
```

- [Name] Specifies the name of the experiment.
- [Run Parameters] Specifies a fix directory name where you can create and define keys which can be presented at Run start for run condition selection. The actual activation of any of those line is done via a "logical link key" defined in the Edit on Start/ sub-tree. The links don't have to point to run parameters necessarily. They can point to any ODB key including the logger settings. It can make sense to create a link to the logger setting which enables/disables writing of data. A quick test run can then be made without data logging for example:

```
[local]/>create key "/Experiment/Run parameters"
```

Then one or more run parameters can be created in that directory,

```
[local]Run parameters>create int "Run mode"
[local]Run parameters>create string Comment
```

- [Edit on Start] Specifies a fix directory name where you can define an ODB link (similar to a symbolic link in UNIX) key to the pre-defined directory Run Parameters. Any link key present in this directory pointing to a valid ODB key will be requested for input during the run start procedure.

  A new feature has been added to this section for the possibility of preventing the user to change the run number from the web interface during the start sequence. By defining the key **/Experiment/Edit** on Start/Edit run number as a boolean variable the ability of editing the run number is enabled or disabled. By default if this key is not present the run number is editable.

```
[local]/>create key "Experiment/Edit on start"
[local]/>cd "Experiment/Edit on start"
[local]/>ln "/Experiment/Run parameters/Run mode" "Run mode"
```

  When a run is started from ODBEdit, all links in /Experiment/Edit on start are scanned and read in:

```
[local]/>start
Run mode [0]:1
Run number [3]:<return to accept>
Are the above parameters correct?
([y]/n/q): <return to accept "y">
Starting run #2
Run #2 started


[local]/>cd "Experiment/Edit on start"
[local]/>create BOOL "Edit run number"
```

- [Lock when running] Specifies a fix directory for defining logical link keys to be set in Read only access mode while the run is in progress. The lock when running can contains logical link to key(s) for setting these keys protection to "read only" while running. In the example below, all the parameters under the declared tree will be switched to read only preventing any parameters modification during the run.

```
[local]/>create key "Experiment/Lock when running"
[local]/>cd "Experiment/Lock when running"
[local]/>ln "/Experiment/Run parameters" "Run parameter"
[local]/>ln "/Logger/Write Data" "Write Data?"
```

- [Security] Specifies a fix directory name where information regarding security can be setup. By default, there is no restriction for user to connect locally or remotely to a given experiment. If an access restriction has to be setup in order to protect the experiment from unwilling access, a password mechanism has to be defined. This directory is automatically created when the command **passwd** is issued in ODB (see below).

- [Password] Specifies the encrypted password for accessing current experiment.

```
[local]/>passwd
Password:<xxxx>
Retype password:<xxxx>
```

To remove the full password checking mechanism, the ODB security sub-tree has to be entirely deleted using the following command:

```
[local]/>rm /Experiment/Security
Are you sure to delete the key
"/Experiment/Security"
and all its subkeys? (y/[n]) y
```

After running the odb command passwd, four new sub-fields will be present under the Security tree.

- Password
- Allowed hosts
- Allowed programs
- Web Password

- [Allowed hosts] Specifies a fix directory name where allowed remote hostname can be defined for free access to the current experiment. While the access restriction can make sense to deny access to outsider to a given experiment, it can be annoying for the people working directly at the back-end computer or for the automatic frontend reloading mechanism (MS-DOS, VxWorks configuration). To address this problem specific hosts can be exempt from having to supply a password and being granted of full access.

```
[local]/>cd "/Experiment/Security/Allowed hosts"
[local]rhosts>create int myhost.domain
[local]rhosts>
```

Where <myhost>.<domain> has to be replaces by the full IP address of the host requesting full clearance.

- [Allowed programs] Specifies a list of programs having full access to the ODB independently of the node they running from.

```
[local]/>cd "/Experiment/Security/Allowed programs"
[local]:S>create int mstat
[local]:S>
```

- [Web Password] Specifies a separate password for the Web server access (mhttpd task). If this field is active, the user will be requested to provide the "Web Password" when accessing the requested experiment in a "Write Access". In all condition the Read Only Access" is available.

6.14.6 ODB /History Tree

This tree is automatically created when the logger is started. The logger will create a default sub-tree containing the following structure:

```
[local:midas:S]/History>ls -l -r
Key name                        Type    #Val  Size  Last Opn Mode Value
---------------------------------------------------------------------------
History                         DIR
    Links                       DIR
        System                  DIR
    Trigger per sec.    /Equipment/Trigger/Statistics/Events per sec.
    Trigger kB per sec. /Equipment/Trigger/Statistics/kBytes per sec.

[local:midas:S]/>cd /History/Links/System/
[local:midas:S]System>ls -l
Key name            Type  #Val Size Last Opn Mode Value
---------------------------------------------------------
Trigger per sec.    LINK  1    46   >99d 0   RWD  /Equipment/Trigger/Statistics/Events per sec.
Trigger kB per sec. LINK  1    46   >99d 0   RWD  /Equipment/Trigger/Statistics/kBytes per sec.
```

A second sub-tree is added to the /History by the mhttpd task Midas web server when the button "History" on the main status page is pressed.

```
[local:midas:S]/History>ls -l -r Display
Key name                        Type    #Val  Size  Last Opn Mode Value
---------------------------------------------------------------------------
Display                         DIR
  Default                       DIR
    Trigger rate                DIR
        Variables               STRING  2     32    36h  0   RWD
                                        [0]             System:Trigger per sec.
                                        [1]             System:Trigger kB per sec.
        Factor                  FLOAT   2     4     36h  0   RWD
                                        [0]             1
                                        [1]             1
        Timescale               INT     1     4     36h  0   RWD  3600
        Zero ylow               BOOL    1     4     36h  0   RWD  y
```

This define a default history display under the Midas web server as long as the reference to "System" is correct. See History system for more information regarding explanation on these fields.

Where the 2 trigger fields are symbolic links to the given path. The sub-tree **System** defines a "virtual" equipment and get by the system assigned a particular "History Event ID".

6.14.7 ODB /Alarms Tree

This branch contains system information related to alarms. Currently the overall alarm is checked once every minute. Once the alarm has been triggered, the message associated to the alarm can be repeated at a different rate. The structure is split in 2 sections. The **"Alarms"** itself which define the condition to be tested and the **"Classes"** which defines the action to be taken when the alarm occurs. In order to make the system flexible, beside some default message logging (Classes/Write system message), each action may have a particular "detached script" spawned by it (Classes/Execute command).

```
odb -e expt -h host
[host:expt:Stopped]/Alarms>ls -lr
Key name                        Type    #Val  Size  Last Opn Mode Value
--------------------------------------------------------------------------
Alarms                          DIR
    Alarm system active         BOOL    1     4     6h   0   RWD  n
    Alarms                      DIR
        Test                    DIR
            Active              BOOL    1     4     31h  0   RWD  n
            Triggered           INT     1     4     31h  0   RWD  0
            Type                INT     1     4     31h  0   RWD  3
            Check interval      INT     1     4     31h  0   RWD  60
            Checked last        DWORD   1     4     31h  0   RWD  0
            Time triggered firstSTRING  1     32    31h  0   RWD
            Time triggered last STRING  1     32    31h  0   RWD
            Condition           STRING  1     256   31h  0   RWD  /Runinfo/Run number > 10
            Alarm Class         STRING  1     32    31h  0   RWD  Alarm
            Alarm Message       STRING  1     80    31h  0   RWD  Run number became too large
        wc3_anode               DIR
            Active              BOOL    1     4     31h  0   RWD  n
            Triggered           INT     1     4     31h  0   RWD  0
            Type                INT     1     4     31h  0   RWD  3
            Check interval      INT     1     4     31h  0   RWD  10
            Checked last        DWORD   1     4     31h  0   RWD  958070825
            Time triggered firstSTRING  1     32    31h  0   RWD
            Time triggered last STRING  1     32    31h  0   RWD
            Condition           STRING  1     256   31h  0   RWD  /equipment/chv/variables/chvv[6] <
            Alarm Class         STRING  1     32    31h  0   RWD  Alarm
            Alarm Message       STRING  1     80    31h  0   RWD  WC3 Anode voltage is too low
        chaos                   DIR
            Active              BOOL    1     4     31h  0   RWD  n
            Triggered           INT     1     4     31h  0   RWD  0
            Type                INT     1     4     31h  0   RWD  3
            Check interval      INT     1     4     31h  0   RWD  10
            Checked last        DWORD   1     4     31h  0   RWD  0
            Time triggered firstSTRING  1     32    31h  0   RWD
            Time triggered last STRING  1     32    31h  0   RWD
            Condition           STRING  1     256   31h  0   RWD  /Equipment/B12Y/Variables/B12Y[2]
            Alarm Class         STRING  1     32    31h  0   RWD  Alarm
            Alarm Message       STRING  1     80    31h  0   RWD  CHAOS magnet has tripped.
    Classes                     DIR
        Alarm                   DIR
            Write system messageBOOL    1     4     31h  0   RWD  y
            Write Elog message  BOOL    1     4     31h  0   RWD  n
            System message interINT     1     4     31h  0   RWD  60
```

```
        System message last DWORD   1     4     31h  0   RWD  0
        Execute command     STRING  1     256   31h  0   RWD
        Execute interval    INT     1     4     31h  0   RWD  0
        Execute last        DWORD   1     4     31h  0   RWD  0
        Stop run            BOOL    1     4     31h  0   RWD  n
    Warning                 DIR
        Write system messageBOOL    1     4     31h  0   RWD  y
        Write Elog message  BOOL    1     4     31h  0   RWD  n
        System message interINT     1     4     31h  0   RWD  60
        System message last DWORD   1     4     31h  0   RWD  0
        Execute command     STRING  1     256   31h  0   RWD
        Execute interval    INT     1     4     31h  0   RWD  0
        Execute last        DWORD   1     4     31h  0   RWD  0
        Stop run            BOOL    1     4     31h  0   RWD  n
```

- [Alarm system active] Overall Alarm enable flag.
- [Alarms] Sub-tree defining each individual alarm condition.
- [Classes] Sub-tree defining each individual action to be performed by a pre-defined and requested alarm.

6.14.8 ODB /Script Tree

This branch permits to invoke scripts from the web page. By creating the ODB tree **/Script** every entry in that tree will be available on the Web status page with the name of the key. Each key entry is then composed with a list of ODB field (or links). The first ODB field should be the executable command followed by as many arguments as you wish to be passed to the script.

```
[host::expt:Stopped]/Script>ls
BNMR Hold
Continue
Real
Test
Kill
[host:expt:Stopped]/Script>ls -lr Continue
Key name            Type    #Val  Size  Last Opn Mode Value
---------------------------------------------------------
Continue            DIR
    cmd             STRING  1     128   39h  0   RWD /home/bnmr/perl/continue.pl
    Name            STRING  1     32    28s  0   RWD  bnmr1
    hold            BOOL    1     4     31h  0   RWD  n
```

6.14.9 ODB /Alias Tree

This branch is not present until the user creates it. It is meant to contain symbolic links list to any ODB location. It is used for the Midas web interface where all the sub-trees will appear in the main window. By default the clicking of the button in the web interface will spawn a new frame. To force the display of the alias link in the same frame, a "&" has to be added to the name of the alias.

```
odbedit
ls
create key Alias
cd Alias
ln /Equipment/Trigger/Common "Trig Setting" <-- New frame
ln /Equipment/Trigger/Common "Trig Setting&"      <-- Same frame
```

6.14.10 ODB /Elog Tree

This branch describes the Elog settings used through the Midas web server. See mhttpd task for setting up the different Elog page display.

```
[local:midas:S]/Elog>ls -lr
Key name                        Type    #Val  Size  Last Opn Mode Value
--------------------------------------------------------------------------------
Elog                            DIR
    Email                       STRING  1     64    25h  0   RWD  midas@triumf.ca
    Display run number          BOOL    1     4     25h  0   RWD  y
    Allow delete                BOOL    1     4     25h  0   RWD  n
    Types                       STRING  20    32    25h  0   RWD
                                        [0]             Routine
                                        [1]             Shift summary
                                        [2]             Minor error
                                        [3]             Severe error
                                        [4]             Fix
                                        [5]             Question
                                        [6]             Info
                                        [7]             Modification
                                        [8]             Reply
                                        [9]             Alarm
                                        [10]            Test
                                        [11]            Other
                                        [12]
                                        [13]
                                        [14]
                                        [15]
                                        [16]
                                        [17]
                                        [18]
                                        [19]
    Systems                     STRING  20    32    25h  0   RWD
                                        [0]             General
```

```
                                    [1]             DAQ
                                    [2]             Detector
                                    [3]             Electronics
                                    [4]             Target
                                    [5]             Beamline
                                    [6]
                                    [7]
                                    [8]
                                    [9]
                                    [10]
                                    [11]
                                    [12]
                                    [13]
                                    [14]
                                    [15]
                                    [16]
                                    [17]
                                    [18]
                                    [19]
Buttons
                          8h
                          24h
                          3d
                          7d
Host name                 myhost.triumf.ca
    SMTP host             STRING  1     64    25h  0   RWD  trmail.triumf.ca
```

- [Email] Defines the Email address for Elog reply.
- [Display run number] Allows to disable the run number display in the Elog entries.
- [Allow delete] Flag for permiting the deletion of Elog entry.
- [Types] Pre-defined types displayed when composing an Elog entry. A maximum of 20 types are available. The list will be terminated by the encounter of the first blank type.
- [Systems] Pre-defined categories displayed when composing an Elog entry. A maximum of 20 types are available. The list will be terminated by the encounter of the first blank type.
- [SMTP host] Mail server address for routing the composed Elog message to the destination.
- [Buttons] Permits to recall up to four possible time span for the Elog command.
- [Host name] Host name.
- [Email <...>] Email address to where the message should be sent when composing it under "Systems" of the type <...>,

6.14.11 ODB /Programs Tree

System created tree containing task specific characteristics such as the watchdog and alarm condition. See Alarm System .

```
Key name                        Type    #Val  Size  Last Opn Mode Value
------------------------------------------------------------------------
Programs                        DIR
    EBuilder                    DIR
        Required                BOOL    1     4     0s   0   RWD  y
        Watchdog timeout        INT     1     4     0s   0   RWD  10000
        Check interval          DWORD   1     4     0s   0   RWD  10000
        Start command           STRING  1     256   0s   0   RWD  mevb -D
        Auto start              BOOL    1     4     0s   0   RWD  n
        Auto stop               BOOL    1     4     0s   0   RWD  n
        Auto restart            BOOL    1     4     0s   0   RWD  n
        Alarm class             STRING  1     32    0s   0   RWD  Alarm
        First failed            DWORD   1     4     0s   0   RWD  0
```

6.14.12 ODB /Lazy Tree

Backup facility Tree. Created with default parameters on the first activation of lazylogger task. This task connects to a defined channel (i.e: Tape). when started. Multiple instance of the program can run contemporary.

```
Key name                        Type    #Val  Size  Last Opn Mode Value
------------------------------------------------------------------------
Lazy                            DIR
    Tape                        DIR
        Settings                DIR
            Maintain free space(INT     1     4     23h  0   RWD  15
            Stay behind         INT     1     4     23h  0   RWD  -1
            Alarm Class         STRING  1     32    23h  0   RWD
            Running condition   STRING  1     128   23h  0   RWD  ALWAYS
            Data dir            STRING  1     256   23h  0   RWD  /data_onl/current
            Data format         STRING  1     8     23h  0   RWD  YBOS
            Filename format     STRING  1     128   23h  0   RWD  run%05d.ybs
            Backup type         STRING  1     8     23h  0   RWD  Tape
            Execute after rewindSTRING  1     64    23h  0   RWD  ask_for_tape.sh
            Path                STRING  1     128   23h  0   RWD  /dev/nst0
            Capacity (Bytes)    FLOAT   1     4     23h  0   RWD  4.8e+10
            List label          STRING  1     128   3h   0   RWD  tw0078
            Execute before writiSTRING  1     64    23h  0   RWD  lazy_prewrite.csh
            Execute after writinSTRING  1     64    23h  0   RWD  rundb_addrun.pl
        Statistics              DIR
            Backup file         STRING  1     128   3h   0   RWDE run05627.ybs
            File size [Bytes]   FLOAT   1     4     3h   0   RWDE 2.00176e+09
            KBytes copied       FLOAT   1     4     3h   0   RWDE 2.00176e+09
            Total Bytes copied  FLOAT   1     4     3h   0   RWDE 2.00176e+09
            Copy progress [%]   FLOAT   1     4     3h   0   RWDE 100
```

```
            Copy Rate [bytes perFLOAT   1     4     3h   0   RWDE 6.21462e+06
            Backup status [%]   FLOAT   1     4     3h   0   RWDE 4.17034
            Number of Files     INT     1     4     3h   0   RWDE 1
            Current Lazy run    INT     1     4     3h   0   RWDE 5627
        List                    DIR
            TW0076              INT     15    4     3h   0   RWD
                                        [0]             5575
                                        [1]             5576
                                        [2]             5577
```

6.14.13 ODB /EBuilder Tree

The Event Builder tree is created by mevb task and is placed in the Equipment list.

```
Key name                        Type    #Val  Size  Last Opn Mode Value
------------------------------------------------------------------------------
EBuilder                        DIR
    Settings                    DIR
        Event ID                WORD    1     2     65h  0   RWD  1
        Trigger mask            WORD    1     2     65h  0   RWD  1
        Buffer                  STRING  1     32    65h  0   RWD  SYSTEM
        Format                  STRING  1     32    65h  0   RWD  YBOS
        Event mask              DWORD   1     4     65h  0   RWD  3
        hostname                STRING  1     64    3h   0   RWD  myhost
    Statistics                  DIR
        Events sent             DOUBLE  1     8     3h   0   RWD  653423
        Events per sec.         DOUBLE  1     8     3h   0   RWD  1779.17
        kBytes per sec.         DOUBLE  1     8     3h   0   RWD  0
    Channels                    DIR
        Frag1                   DIR
            Settings            DIR
                Event ID        WORD    1     2     65h  0   RWD  1
                Trigger mask    WORD    1     2     65h  0   RWD  65535
                Buffer          STRING  1     32    65h  0   RWD  YBUF1
                Format          STRING  1     32    65h  0   RWD  YBOS
                Event mask      DWORD   1     4     65h  0   RWD  1
            Statistics          DIR
                Events sent     DOUBLE  1     8     3h   0   RWD  653423
                Events per sec. DOUBLE  1     8     3h   0   RWD  1779.17
                kBytes per sec. DOUBLE  1     8     3h   0   RWD  0
        Frag2                   DIR
            Settings            DIR
                Event ID        WORD    1     2     65h  0   RWD  5
                Trigger mask    WORD    1     2     65h  0   RWD  65535
                Buffer          STRING  1     32    65h  0   RWD  YBUF2
                Format          STRING  1     32    65h  0   RWD  YBOS
                Event mask      DWORD   1     4     65h  0   RWD  2
            Statistics          DIR
                Events sent     DOUBLE  1     8     3h   0   RWD  653423
                Events per sec. DOUBLE  1     8     3h   0   RWD  1779.17
                kBytes per sec. DOUBLE  1     8     3h   0   RWD  0
```

6.14.14 ODB /Custom Tree

Web string for custom web page. **Editable** ONLY from your Web browser through Custom page .

```
Key name                          Type    #Val  Size  Last Opn Mode Value
---------------------------------------------------------------------------
WebLtno&                          STRING  1     2976  25h  0   RWD  <multi-line>
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
   <meta name="GENERATOR" content="Mozilla/4.76 [en] (Windows NT 5.0; U) [Netscape]">
   <meta name="Author" content="Pierre-Andr?Amaudruz">
   <title>Set value</title>
</head>
<body text="#000000" bgcolor="#FFFFCC" link="#FF0000" vlink="#800080" alink="#0000FF">
<form method="GET" action="http://myhost.triumf.ca:8081/CS/WebLtno&">
<input type=hidden name=exp value="ltno">
<center><table CELLSPACING=1 CELLPADDING=1 COLS=3 WIDTH="100%" BGCOLOR="#99FF99" >
<caption><b><font face="Georgia"><font color="#000099"><font size=+2>LTNO
Custom Web Page</font></font></font></b></caption>

<tr BGCOLOR="#FFCC99">
<td><b><font color="#FF0000">Actions: </font></b>
<input type=submit name=cmd value=Status>
<input type=submit name=cmd value=Start>
<input type=submit name=cmd value=Stop>
...
<td BGCOLOR="#66FFFF"><b>Polarity section:</b>
<br>Run#: <odb src="/runinfo/run number">
<br>Run#: <odb src="/runinfo/run number">
<br>Run#: <odb src="/runinfo/run number">
<br>Run#: <odb src="/runinfo/run number" edit=1></td>
</tr>
</table></center>

<img src="http://myhost.triumf.ca:8081/HS/Meterdis.gif?exp=ltno&scale=12h&width=400">
<img src="http://myhost.triumf.ca:8081/HS/Bridge.gif?exp=ltno&scale=12h&width=400">
<b><i><font color="#000099"><a href="http://myhost.triumf.ca/ltno/index.html">
<br>
LTNO help</a></font></i></b>
</body>
</html>
```

6.14.15 Hot Link

It is often desirable to modify hardware parameters like discriminator levels or trigger logic connected to the frontend computer. Given the according hardware is accessible from the frontend code, theses parameters are easily controllable when a hot-link ODB is established between the frontend and the ODB itself.

HotLink process

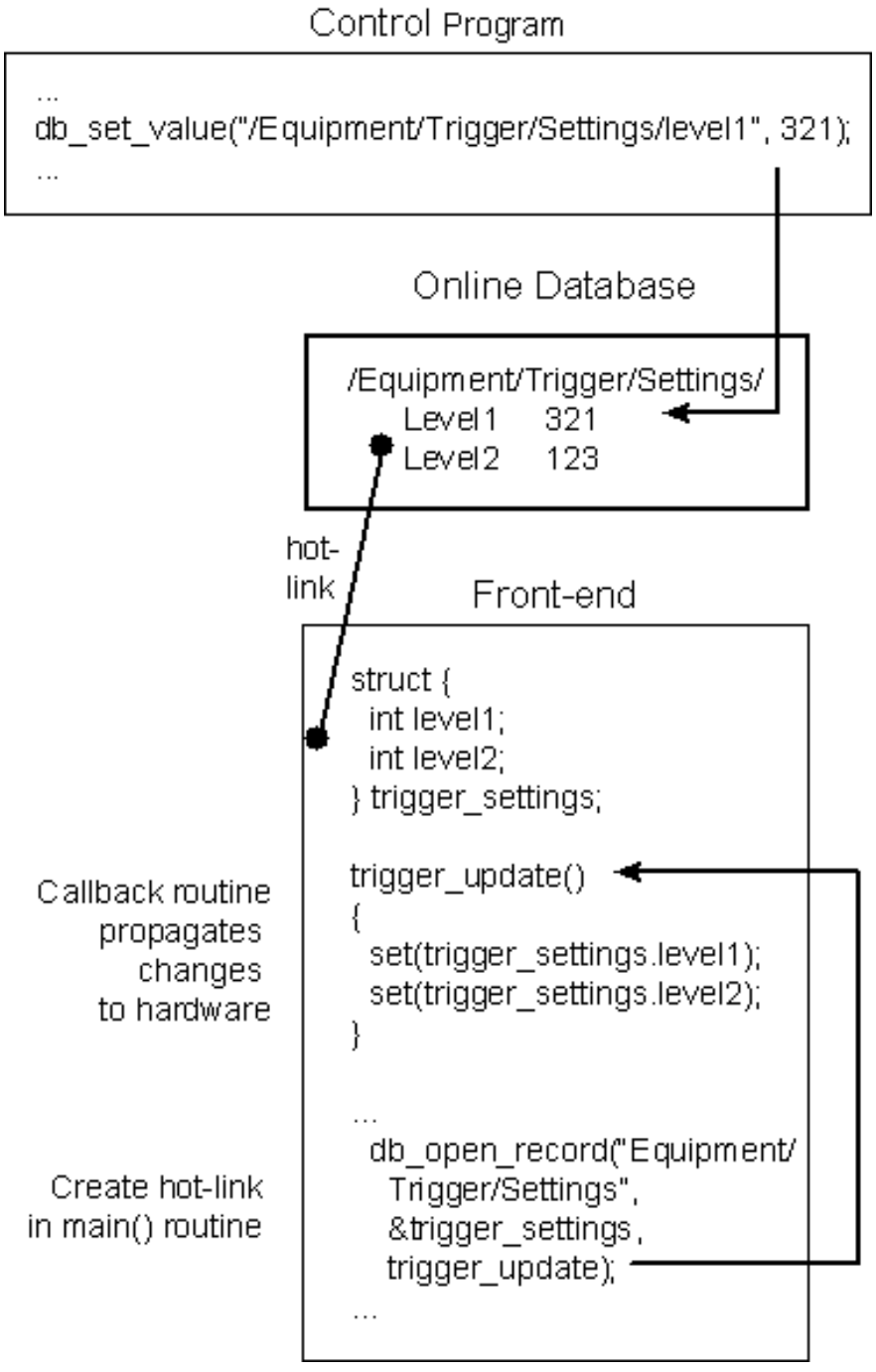


Figure 34: HotLink process

First the parameters have to be defined in the ODB under the Settings tree for the given equipment. Let's assume we have two discriminator levels belonging to the trigger electronics, which should be controllable. Following commands define these levels in the ODB:

```
[local]/>cd /Equipment/Trigger/
[local]Trigger>create key Settings
[local]Trigger>cd Settings
[local]Settings>create int level1
[local]Settings>create int level2
[local]Settings>ls
```

The frontend can now map a C structure to these settings. In order to simplify this process, ODBEdit can be requested to generate a header file containing this C structure. This file is usually called event.h. It can be generated in the current directory with the ODB command **make** which generates in the current directory the header file **experim.h** :

```
[local]Settings>make
```

Now this file can be copied to the frontend directory and included in the frontend source code. It contains a section with a C structure of the trigger settings and an ASCII representation:

```
typedef struct {
  INT       level1;
  INT       level2;
  TRIGGER_SETTINGS;

#define TRIGGER_SETTINGS_STR(_name) char *_name[] = {\
"[.]",\
"level1 = INT : 0",\
"level2 = INT : 0",\
"",\
NULL
```

This definition can be used to define a C structure containing the parameters in frontend.c:

```
#include <experim.h>

TRIGGER_SETTINGS trigger_settings;
```

A hot-link between the ODB values and the C structure is established in the frontend_init() routine:

```
INT frontend_init()
{
```

```
HNDLE hDB, hkey;
TRIGGER_SETTINGS_STR(trigger_settings_str);

  cm_get_experiment_database(&hDB, NULL);

  db_create_record(hDB, 0,
    "/Equipment/Trigger/Settings",
    strcomb(trigger_settings_str));

  db_find_key(hDB, 0,
    "/Equipment/Trigger/Settings", &hkey);

  if (db_open_record(hDB, hkey,
      &trigger_settings,
      sizeof(trigger_settings), MODE_READ,
      trigger_update) != DB_SUCCESS)
    {
    cm_msg(MERROR, "frontend_init",
      "Cannot open Trigger Settings in ODB");
    return -1;

  return SUCCESS;
```

The db_create_record() function re-creates the settings sub-tree in the ODB from the ASCII representation in case it has been corrupted or deleted. The db_open_record() now establishes the hot-link between the settings in the ODB and the trigger_settings structure. Each time the ODB settings are modified, the changes are written to the trigger_settings structure and the callback routine trigger_update() is executed afterwards. This routine has the task to set the hardware according to the settings in the trigger_settings structure.

It may look like:

```
void trigger_update(INT hDB, INT hkey)
{
  printf("New levels: %d %d",
    trigger_settings.level1,
    trigger_settings.level2);
```

Of course the printf() function should be replaced by a function which accesses the hardware properly. Modifying the trigger values with ODBEdit can test the whole scheme:

```
[local]/>cd /Equipment/Trigger/Settings
[local]Settings>set level1 123
[local]Settings>set level2 456
```

Immediately after each modification the frontend should display the new values. The settings can be saved to a file and loaded back later:

```
[local]/>cd /Equipment/Trigger/Settings
```

```
[local]Settings>save settings.odb
[local]Settings>set level1 789
[local]Settings>load settings.odb
```

The settings can also be modified from any application just by accessing the ODB. Following listing is a complete user application that modifies the trigger level:

```
#include <midas.h>

main()
{
HNDLE hDB;
INT   level;

  cm_connect_experiment("", "Sample", "Test",
                        NULL);
  cm_get_experiment_database(&hDB, NULL);

  level = 321;
  db_set_value(hDB, 0,
    "/Equipment/Trigger/Settings/Level1",
    &level, sizeof(INT), 1, TID_INT);

  cm_disconnect_experiment();
```

The following figure summarizes the involved components:

To make sure a hot-link exists, one can use the ODBEdit command **sor** (show open records):

```
 [local]Settings>cd /
[local]/>sor
/Equipment/Trigger/Settings open 1 times by ...
```

6.14.16 History system

The history system is an add-on capability build in the data logger (see mlogger task) to record information in parallel to the data logging. This information is recorded with a time stamp and saved into "data base file" like for later retrieval. One set of file is created per day containing all the requested history events.

The history is working only if the logger is running, but it is not necessary to have any channel enabled.

The definition of the history event is done through two different means:

- **frontend** history event: Each equipment has the capability to generate "history data" if the particular history field value is different then zero. The value will reflect the periodicity of the history logging (see The Equipment structure).

- **"Virtual History event"**: Composed within the Online Database under the specific tree "/History/Links" (see ODB /History Tree)

Both definition mode takes effects when the data logger gets a "start run" transition. Any modification during the run is not applied until the next run is started.

- [frontend history event] As mentioned earlier, each equipment can be enabled to generate history event based on the periodicity of the history value (in second!). The content if the event will be completely copied into the history files using the definition of the event as tag names for every element of the event.

The history variable name for each element of the event is composed following one of the rules below:

- [bank event] **/equipment/**<...>/Variables/<bank name>[] is the only reference of the event, the history name is composed of the bank name follwed by the corresponding index of the element.
- [bank event] **/equipment/**<...>/Settings/Names <bank_name>[] is present, the history name is composed of the corresponding name found in the "Names <bank_name>" array. The size of this array should match the size of the **/equipment/**<...>/Variables/<bank name[]> .

```
[host:chaos:Running]Target>ls -l -r
Key name                        Type    #Val  Size  Last Opn Mode Value
---------------------------------------------------------------------------
Target                          DIR
    settings                    DIR
        Names TGT_              STRING  7     32    10h  0   RWD
                                        [0]             Time
                                        [1]             Cryostat vacuum
                                        [2]             Heat Pipe pressure
                                        [3]             Target pressure
                                        [4]             Target temperature
                                        [5]             Shield temperature
                                        [6]             Diode temperature
    Common                      DIR
          ...
    Variables                   DIR
        TGT_                    FLOAT   7     4     10s  0   RWD
                                        [0]             114059
                                        [1]             4.661
                                        [2]             23.16
                                        [3]             -0.498
                                        [4]             22.888
                                        [5]             82.099
                                        [6]             40
    Statistics                  DIR
          ...
```

- [fixed event] The names of the individual element under **/equipment/**<...>/variables/ will be used for the history name composition.
- [fixed event with array] If the **/equipment/**<...>/Settings/Names[] exists, each element of the array will be referenced using the corresponding name of the **/Settings/Names**[] array.
- [ODB history event]

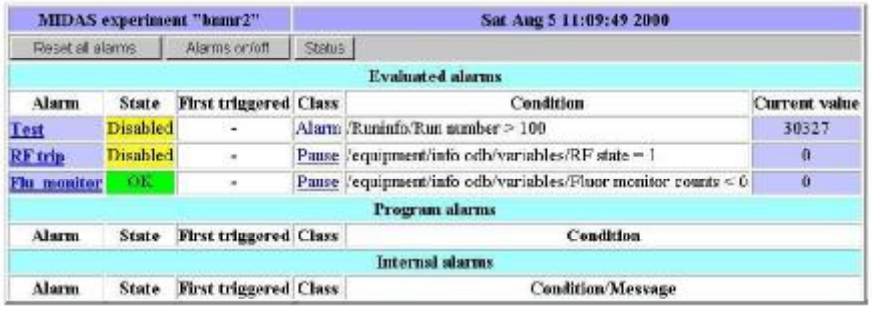
6.14.17 Alarm System

The alarm system is built in and part of the main experiment scheduler. This means no separate task is necessary to beneficate from it, but this feature is active during **ONLINE** mode **ONLY** . Alarm setup and activation is done through the Online DataBase. Alarm system includes several other features such as: sequencing control of the experiment. The alarm capabilities are:

- Alarm setting on any ODB variables against threshold parameter.
- Alarm check frequency
- Alarm trigger frequency
- Customizable alarm scheme, under this scheme multiple choice of alarm type can be selected.
- Program control on run transition.

Beside the setup through ODBEdit, the Alarm can also be setup through the Midas web page..

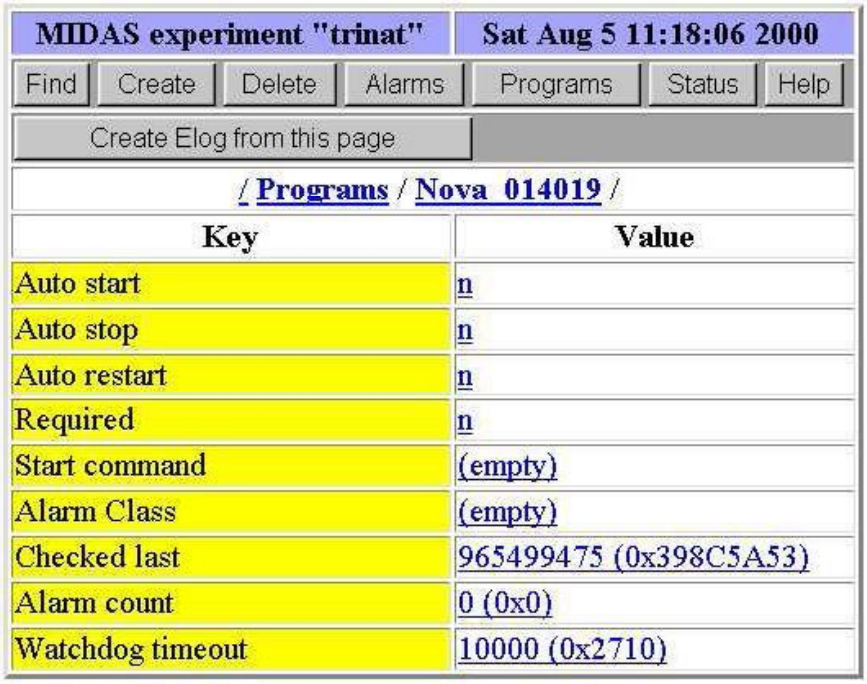
Midas Web Alarm setting display



MIDAS experiment "bnmr2"				Sat Aug 5 11:09:49 2000	
Reset all alarms	Alarms on/off	Status			
Evaluated alarms					
Alarm	**State**	**First triggered**	**Class**	**Condition**	**Current value**
Test	Disabled	-	Alarm	/Runinfo/Run number > 100	30327
RF trip	Disabled	-	Pause	/equipment/info odb/variables/RF state = 1	0
Fluo_monitor	OK	-	Pause	/equipment/info odb/variables/Fluor monitor counts < 0	0
Program alarms					
Alarm	**State**	**First triggered**	**Class**	**Condition**	
Internal alarms					
Alarm	**State**	**First triggered**	**Class**	**Condition/Message**	

Figure 35: Midas Web Alarm setting display

Midas Web Alarm setting display



MIDAS experiment "trinat"	Sat Aug 5 11:18:06 2000
Find Create Delete Alarms Programs Status Help	
Create Elog from this page	
/ Programs / Nova_014019 /	
Key	**Value**
Auto start	n
Auto stop	n
Auto restart	n
Required	n
Start command	(empty)
Alarm Class	(empty)
Checked last	965499475 (0x398C5A53)
Alarm count	0 (0x0)
Watchdog timeout	10000 (0x2710)

Figure 36: Midas Web Alarm setting display

Midas Web Alarm Program status display

Figure 37: Midas Web Alarm Program status display

Internal features - Top - Data format

6.15 Quick Start

Components - Top - Internal features

This section is under revision to better reflect the latest installation and basic operation of the Midas package.

... This section will... describes step-by-step the installation procedure of the Midas package on several platform as well as the procedure to run a demo sample experiment. In a second stage, the frontend or the analyzer can be moved to another computer to test the remote connection capability.

The Midas Package source and binaries can be found at : `PSI` or at `TRIUMF` . An `online SVN web site` is also available for the latest developments.

Even though Midas is available for multiple platforms, the following description are for Linux installation and Windows installation.

6.15.1 Linux installation

1. **Extraction:**
   - **Compressed files** The compressed file contains the source and binary code. It does expand under the directory name of **midas**. This extraction can be done at the user level.

```
cd /home/mydir
tar -zxvf midas-1.9.x.tar.gz
```

The midas directory structure will be composed of several subdirectories such as:

```
>ls
COPYING  doc/      examples/  include/  linux/     makefile.nt  mscb/  utils/
CVS/     drivers/  gui/       java/     Makefile*  mcleanup*    src/   vxworks/
```

- **RPM** Current RPM is not fully up-to-date. We suggest that you use the compressed files or the **SVN repository**. In the case of the **rpm**, the binaries are placed in the /usr/local/bin , /usr/local/include, /usr/local/lib.
- **SVN** The source code can be extracted from the SVN repository. An anonymous access is available under the username svn and password svn which may be required several time. SVN provides also a quick **tarball** creation within the web interface!

```
svn co svn+ssh://svn@savannah.psi.ch/afs/psi.ch/project/meg/svn/midas/trunk midas
svn co svn+ssh://svn@savannah.psi.ch/afs/psi.ch/project/meg/svn/mxml/trunk mxml
```

If you expect to run the ROME analyzer you can extract the SVN package following the same procedure.

```
svn co svn+ssh://svn@savannah.psi.ch/afs/psi.ch/project/meg/svn/rome/trunk rome
```

For the Histogram display tool ROODY the package still resides under CVS but will be soon moved to SVN.

```
cvs -d anoncvs@midas.triumf.ca:/usr/local/cvsroot checkout roody
```

2. **Installation:** The installation consists in placing the image files in the **/usr/local/** directories. This operation requires superuser privilege. The open "install" from the Makefile will automatically do this installation for you.

```
cd /home/mydir/midas
su -
make install
```

3. **Configuration:** Several system files needs to be modified for the full Midas implementation.
   - **/etc/services :** For remote access. Inclusion of the midas service. Add following line:

```
# midas service
midas           1175/tcp                        # Midas server
```

   - **/etc/xinetd.d/midas :** Daemon definition. Create new file named **midas**

```
service midas
{
        flags                   = REUSE NOLIBWRAP
        socket_type             = stream
        wait                    = no
        user                    = root
```

```
        server                  = /usr/local/bin/mserver
        log_on_success          += USERID HOST PID
        log_on_failure          += USERID HOST
        disable                 = no
}
```

- **/etc/ld.so.conf :** Dynamic Linked library list. Add directory pointing to location of the midas.so file (add /usr/local/lib).

  ```
  /usr/local/lib
  ```

  The system is now build by default in static which prevent to have to setup the .so path through either the environment **LD_LIBRARY_PATH** or the ld.so.conf.

- **/etc/exptab :** Midas Experiment definition file (see below).

4. **Experiment definition:** Midas system supports multiple experiment running contemporary on the same computer. Even though it may not be efficient, this capability makes sense when the experiments are simple detector lab setups which shared hardware resources for data collection. In order to support this feature, Midas requires a uniquely identified set of parameter for each experiment that is used to define the location of the Online Database.

   Every experiment under Midas has its own ODB. In order to differentiate them, an experiment name and directory are assigned to each experiment. This allows several experiments to run concurrently on the same host using a common Midas installation.

   Whenever a program participating in an experiment is started, the experiment name can be specified as a command line argument or as an environment variable.

   A list of all possible running experiments on a given machine is kept in the file **exptab**. This file **exptab** is expected by default to be located under **/etc/exptab**. This can be overwritten by the Environment variables MIDAS_EXPTAB.

   exptab file is composed of one line per experiment definition. Each line contains three parameters, i.e: **experiment name**, **experiment directory name** and **user name**. Example:

   ```
   #
   # Midas experiment list
   midas   /home/midas/online     midas
   decay   /home/slave/decay_daq  slave
   ```

   Experiments not defined into **exptab** are not accessible remotely, but can still be accessed locally using the Environment variables MIDAS_DIR if defined. This environment superceed the **exptab** definition.

5. **Compilation & Build:** You should be able to rebuild the full package once the Midas tree structure has been placed in your temporary directory. The compilation and link will try to generate the **rmidas** application which requires **ROOT**.

The application **mana** will also be compiled for **HBOOK** and **ROOT**. Look in the make listing below for the HAVE_HBOOK, HAVE_ROOT.

```
> cd /home/mydir/midas
> make
cc -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/lib/midas.o src/midas.c
cc -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/lib/system.o src/system.c
cc -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/lib/mrpc.o src/mrpc.c
cc -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/lib/odb.o src/odb.c
cc -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/lib/ybos.o src/ybos.c
cc -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/lib/ftplib.o src/ftplib.c
rm -f linux/lib/libmidas.a
ar -crv linux/lib/libmidas.a linux/lib/midas.o linux/lib/system.o linux/lib/mrpc.o
linux/lib/odb.o linux/lib/ybos.o linux/lib/ftplib.o
a - linux/lib/midas.o
a - linux/lib/system.o
a - linux/lib/mrpc.o
a - linux/lib/odb.o
a - linux/lib/ybos.o
a - linux/lib/ftplib.o
rm -f linux/lib/libmidas.so
ld -shared -o linux/lib/libmidas.so linux/lib/midas.o linux/lib/system.o
linux/lib/mrpc.o linux/lib/odb.o linux/lib/ybos.o linux/lib/ftplib.o -lutil
-lpthread -lc
cc -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/lib/mana.o src/mana.c
cc -Dextname -DHAVE_HBOOK -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib
-DINCLUDE_FTPLIB -DOS_LINUX -fPIC    -o linux/lib/hmana.o src/mana.c
...
g++ -DHAVE_ROOT -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB
-DOS_LINUX -fPIC   -D_REENTRANT -I/home1/midas/ root/include -o linux/lib/rmana.o
src/mana.c
g++ -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINU
-fPIC   -o linux/lib/mfe.o src/mfe.c
cc -Dextname -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib
-DINCLUDE_FTPLIB -DOS_LINUX -fPIC    -o linux/lib/fal.o src/fal.c
...
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mserver src/mserver.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mhttpd src/mhttpd.c src/mgd.c -lmidas -lutil -lpthread -lm
g++ -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-DHAVE_ROOT -D_REENTRANT -I/home1/midas/root/include
-o linux/bin/mlogger src/mlogger.c -lmidas
-L/home1/midas/root/lib -lCore -lCint -lHist -lGraf -lGraf3d -lGpad -lTree
-lRint -lPostscript -lMatrix -lPhysics -lpthread -lm -ldl -rdynamic -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/odbedit src/odbedit.c src/cmdedit.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mtape utils/mtape.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
```

```
-o linux/bin/mhist utils/mhist.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mstat utils/mstat.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mcnaf utils/mcnaf.c drivers/bus/camacrpc.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mdump utils/mdump.c -lmidas -lz -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/lazylogger src/lazylogger.c -lmidas -lz -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mchart utils/mchart.c -lmidas -lutil -lpthread
cp -f utils/stripchart.tcl linux/bin/.
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/webpaw utils/webpaw.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/odbhist utils/odbhist.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/melog utils/melog.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mlxspeaker utils/mlxspeaker.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/dio utils/dio.c -lmidas -lutil-lpthread
g++ -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-DHAVE_ROOT -D_REENTRANT -I/home1/midas/root/include -o linux/bin/rmidas src/rmidas.c
-lmidas -L/home1/midas/root/lib -lCore -lCint -lHist -lGraf -lGraf3d -lGpad
-lTree -lRint -lPostscript -lMatrix -lPhysics -lGui -lpthread -lm -ldl -rdynamic
-lutil -lpthread
```

6. **Demo examples:** The midas file structure contains examples of code which can be (should be) used for template. In the **midas/examples/experiment** you will find a full set for frontend and analysis code. The building of this example is performed with the **Makefile** of this directory. The reference to the Midas package is done relative to your current location (../../include). In the case the content of this directory is copied to a different location (template), you will need to modify the local parameters within the Makefile

```
#--------------------------------------------------------------------
# The following lines define direcories. Adjust if necessary
#
DRV_DIR   = ../../drivers/bus
INC_DIR   = ../../include
LIB_DIR   = ../../linux/lib
```

Replace by:

```
#--------------------------------------------------------------------
# The following lines define direcories. Adjust if necessary
#
DRV_DIR   = /home/mydir/midas/drivers/bus
INC_DIR   = /usr/local/include
LIB_DIR   = /usr/local//lib
```

```
> cd /home/mydir/midas/examples/experiment
```

```
  > make
  gcc -g -O2 -Wall -g -I../../include -I../../drivers/bus -DOS_LINUX -Dextname -c
  -o camacnul.o ../../drivers/bus/camacnul.c
  g++ -g -O2 -Wall -g -I../../include -I../../drivers/bus -DOS_LINUX -Dextname -o
  frontend frontend.c
  camacnul.o ../../linux/lib/mfe.o ../../linux/lib/libmidas.a  -lm -lz -lutil
  -lnsl -lpthread
  g++  -D_REENTRANT -I/home1/midas/root/include -DHAVE_ROOT -g -O2 -Wall -g
  -I../../include -I../../drivers/bus -DOS_LINUX -Dextname -o analyzer.o
  -c analyzer.c
  g++  -D_REENTRANT -I/home1/midas/root/include -DHAVE_ROOT -g -O2 -Wall -g
  -I../../include -I../../drivers/bus -DOS_LINUX -Dextname -o adccalib.o -c adccalib.c
  g++  -D_REENTRANT -I/home1/midas/root/include -DHAVE_ROOT -g -O2 -Wall -g
  -I../../include -I../../drivers/bus -DOS_LINUX -Dextname -o adcsum.o -c adcsum.c
  g++  -D_REENTRANT -I/home1/midas/root/include -DHAVE_ROOT -g -O2 -Wall -g
  -I../../include -I../../drivers/bus -DOS_LINUX -Dextname -o scaler.o -c scaler.c
  g++  -o analyzer ../../linux/lib/rmana.o analyzer.o adccalib.o adcsum.o scaler.o
  ../../linux/lib/libmidas.a  -L/home1/midas/root/lib -lCore -lCint -lHist -lGraf
  -lGraf3d -lGpad -lTree -lRint -lPostscript -lMatrix -lPhysics -lpthread -lm -ldl
  -rdynamic -lThread -lm -lz -lutil -lnsl -lpthread
>
```

For testing the system, you can start the frontend as follow:

```
> frontend
  Event buffer size      :     100000
  Buffer allocation      : 2 x 100000
  System max event size  :     524288
  User max event size    :     10000
  User max frag. size    :     5242880
  # of events per buffer :     10

  Connect to experiment ...Available experiments on local computer:
  0 : midas
  1 : root
  Select number:0                    <---- predefined experiment from exptab file

  Sample Frontend connected to <local>. Press "!" to exit                17:27:47
  ================================================================================
  Run status:   Stopped    Run number 0
  ================================================================================
  Equipment     Status     Events     Events/sec Rate[kB/s] ODB->FE    FE->ODB
  --------------------------------------------------------------------------------
  Trigger       OK         0          0.0        0.0        0          0
  Scaler        OK         0          0.0        0.0        0          0
```

In a different terminal window

```
  >odbedit
  Available experiments on local computer:
  0 : midas
  1 : root
  Select number: 0
  [local:midas:S]/>start now
  Starting run #1
  17:28:58 [ODBEdit] Run #1 started
  [local:midas:R]/>
```

The run has been started as seen in the frontend terminal window. See the /examples/experiment/frontend.c for data generation code.

```
Sample Frontend connected to <local>. Press "!" to exit                17:29:07
================================================================================
Run status:   Running    Run number 1
================================================================================
Equipment     Status     Events     Events/sec Rate[kB/s] ODB->FE    FE->ODB
--------------------------------------------------------------------------------
Trigger       OK         865        99.3       5.4        0          9
Scaler        OK         1          0.0        0.0        0          1
```

6.15.2 Windows installation

1. **Extraction:**

2. **Installation:**

3. **Configuration:**

4. **Experiment definition:**

5. **Compilation:**

6. **Demo examples:**

Components - Top - Internal features Internal features - Top - Data format

The Midas system provides several off-the-shelf programs to control, monitor, debug the data aquisition system. Starting with the main utility (odbedit) which provide access to the Online data base and run control.

- odbedit task : Online Database Editor
  - ODB Structure

- Midas Frontend application : Midas Frontend application
- mstat task : Midas ASCII status report
- analyzer task : Midas data analyzer
  - MIDAS Analyzer
- mlogger task : Midas data logger
- lazylogger task : Background data logger
- mdump task : Event dump application
- mevb task : Event Builder application
- mspeaker, mlxspeaker tasks : Speech synthesizer
- mcnaf task : CAMAC standalone application
- mhttpd task : Midas Web server
- melog task : Electronic entry application
- mhist task : History retrieval application
- mchart task : Standalone Chart display application
- mtape task : Tape device manipulator
- dio task : Direct IO provider
- stripchart.tcl file : Tcl/Tk for chart display
- rmidas task : Root/Midas Simple GUI application
- hvedit task : High Voltage Slow Control GUI
- Midas Remote server : Midas Remote server

6.15.3 Midas Frontend application

The purpose of the Midas Frontend application is to collect data from hardware and transmit this information to a central place where data logging and analysis can be performed. This task is achieved with a) a specific code written by the user describing the sequence of action to acquire the hardware data and b) a framework code handling the data flow control, data transmission and run control operation. From Midas version 1.9.5 a new argument (-i index) has been introduced to facilitate the multiple frontend configuration operation required for the Event Builder Functions.

- **Arguments**
  - [-h] : help
  - [-h hostname] : host name (see odbedit task)
  - [-e exptname] : experiment name (see odbedit task)
  - [-D] : Become a Daemon.
  - [-O] : Become a Daemon but keep stdout
  - [-d] : Used for debugging.
  - [-i index] : Set frontend index (used with mevb task).
- **Usage**

6.15.4 odbedit task

odbedit refers to the Online DataBase Editor. This is the main application to interact with the different components of the Midas system.

See ODB Structure for more information.

- **Arguments**
  - [-h] : help.
  - [-h hostname] :Specifies host to connect to. Must be a valid IP host name. This option supersedes the MIDAS_SERVER_HOST environment variable.
  - [-e exptname] :Specifies the experiment to connect to. This option supersedes the MIDAS_EXPT_NAME environment variable.
  - [-c command] :Perform a single command. Can be used to perform operations in script files.
  - [-c @commandFile] :Perform commands in sequence found in the commandFile.
  - [-s size] : size in byte (for creation). Specify the size of the ODB file to be created when no shared file is present in the experiment directory (default 128KB).
  - [-d ODB tree] :Specify the initial entry ODB path to go to.
- **Usage** ODBedit is the MIDAS run control program. It has a simple command line interface with command line editing similar to the UNIX tcsh shell. Following edit keys are implemented:

- [Backspace] Erase the character left from cursor
- [Delete/Ctrl-D] Erase the character under cursor
- [Ctrl-W/Ctrl-U] Erase the current line
- [Ctrl-K] Erase the line from cursor to end
- [Left arrow/Ctrl-B] Move cursor left
- [Right arrow/Ctrl-F] Move cursor right
- [Home/Ctrl-A] Move cursor to beginning of line
- [End/Ctrl-E] Move cursor to end of line
- [Up arrow/Ctrl-P] Recall previous command
- [Down arrow/Ctrl-N] Recall next command
- [Ctrl-F] Find most recent command which starts with current line
- [Tab/Ctrl-I] Complete directory. The command **ls** /Sy <tab> yields to **ls** /System.

- **Remarks**
  - ODBedit treats the hierarchical online database very much like a file system. Most commands are similar to UNIX file commands like ls, cd, chmod, ln etc. The help command displays a short description of all commands.
  - From Midas version 1.9.5, the ODB content can be saved into XML format if the file extension is .xml

```
C:\odbedit
[local:midas:S]/>save odb.xml
[local:midas:S]/>q
more odb.xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- created by ODBEdit on Wed Oct 06 22:48:26 2004 -->
<dir name="root">
  <dir name="System">
    <dir name="Clients">
      <dir name="3880">
        <key name="Name" type="STRING" size="32">ebfe01</key>
        <key name="Host" type="STRING" size="256">pierre2</key>
        <key name="Hardware type" type="INT">42</key>
     <key name="Server Port" type="INT">4658</key>
...
```

```
[local:midas:Stopped]/>help
Database commands ([] are options, <> are placeholders):

alarm                  - reset all alarms
cd <dir>               - change current directory
chat                   - enter chat mode
chmod <mode> <key>     - change access mode of a key
                         1=read | 2=write | 4=delete
```

```
cleanup                 - delete hanging clients
copy <src> <dest>       - copy a subtree to a new location
create <type> <key>     - create a key of a certain type
create <type> <key>[n]  - create an array of size [n]
del/rm [-l] [-f] \<key>  - delete a key and its subkeys
  -l                      follow links
  -f                      force deletion without asking
exec <key>/<cmd>        - execute shell command (stored in key) on server
find <pattern>          - find a key with wildcard pattern
help/? [command]        - print this help [for a specific command]
hi [analyzer] [id]      - tell analyzer to clear histos
ln <source> <linkname>  - create a link to <source> key
load <file>             - load database from .ODB file at current position
ls/dir [-lhvrp] [<pat>] - show database entries which match pattern
  -l                      detailed info
  -h                      hex format
  -v                      only value
  -r                      show database entries recursively
  -p                      pause between screens
make [analyzer name]    - create experim.h
mem                     - show memeory <b> Usage </b>
mkdir <subdir>          - make new <subdir>
move <key> [top/bottom/[n]] - move key to position in keylist
msg [user] <msg>        - compose user message
old                     - display old messages
passwd                  - change MIDAS password
pause                   - pause current run
pwd                     - show current directory
resume                  - resume current run
rename <old> <new>      - rename key
rewind [channel]        - rewind tapes in logger
save [-c -s] <file>     - save database at current position
                          in ASCII format
  -c                      as a C structure
  -s                      as a #define'd string
set <key> <value>       - set the value of a key
set <key>[i] <value>    - set the value of index i
set <key>[*] <value>    - set the value of all indices of a key
set <key>[i..j] <value> - set the value of all indices i..j
scl [-w]                - show all active clients [with watchdog info]
shutdown <client>/all   - shutdown individual or all clients
sor                     - show open records in current subtree
start [number] [now] [-v]   - start a run [with a specific number], [without question]
                             [-v verbose the transaction to the different clients]
stop [-v]                    - stop current run
                             [-v verbose the transaction to the different clients]
trunc <key> <index>     - truncate key to [index] values
ver                     - show MIDAS library version
webpasswd               - change WWW password for mhttpd
wait <key>              - wait for key to get modified
quit/exit               - exit
```

- **Example**

```
>odbedit -c stop
>odbedit
 [hostxxx:exptxxx:Running]/> ls /equipment/trigger
```

6.15.5 mstat task

mstat is a simple ASCII status display. It presents in a compact form the most valuable information of the current condition of the Midas Acquisition system. The display is composed at the most of 5 sections depending on the current status of the experiment. The section displayed in order from top to bottom refer to:

- Run information.
- Equipment listing and statistics if any frontend is active.
- Logger information and statistics if mlogger is active.
- Lazylogger status if lazylogger is active.
- Client listing.

- **Arguments**
  - [-h] : help
  - [-h hostname] : host name (see odbedit task)
  - [-e exptname] : experiment name (see odbedit task)
  - [-l] : loop. Forces mstat to remain in a display loop. Enter "!" to terminate the command.
  - [-w time] : refresh rate in second. Specifies the delay in second before refreshing the screen with up to date information. Default: 5 seconds. Has to be used in conjunction with -l switch. Enter "R" to refresh screen on next update.

- **Usage**

```
 >mstat -l
*-v1.8.0- MIDAS status page ------------------------Mon Apr  3 11:52:52 2000-*
Experiment:chaos       Run#:8699    State:Running          Run time :00:11:34
Start time:Mon Apr  3 11:41:18 2000

FE Equip.   Node              Event Taken    Event Rate[/s] Data Rate[Kb/s]
B12Y        pcch02            67             0.0            0.0
CUM_Scaler  vwchaos           23             0.2            0.2
CHV         pcch02            68             0.0            0.0
KOS_Scalers vwchaos           330            0.4            0.6
KOS_Trigger vwchaos           434226         652.4          408.3
KOS_File    vwchaos           0              0.0            0.0
Target      pcch02            66             0.0            0.0

Logger Data dir: /scr0/spring2000           Message File: midas.log
```

```
Chan.    Active Type      Filename            Events Taken    KBytes Taken
  0      Yes    Disk      run08699.ybs        434206           4.24e+06

Lazy Label      Progress  File name           #files          Total
cni-53          100[%]    run08696.ybs        15              44.3[%]

Clients:  MStatus/koslx0          Logger/koslx0          Lazy_Tape/koslx0
          CHV/pcch02              MChart1/umelba         ODBEdit/koslx0
          CHAOS/vwchaos           ecl/koslx0             Speaker/koslx0
          MChart/umelba           targetFE/pcch02        HV_MONITOR/umelba
          SUSIYBOS/koslx0         History/kosal2         MStatus1/dasdevpc
*-------------------------------------------------------------------------------*
```

6.15.6 analyzer task

analyzer is the main online / offline event analysis application. **analyzer** uses fully the **ODB** capabilities as all the analyzer parameters are dynamically controllable from the Online Database editor odbedit task.

For more detailed information see MIDAS Analyzer

- **Arguments**
  - -c <filename1> <filename2> Configuration file name(s). May contain a '%05d' to be replaced by the run number. Up to ten files can be specified in one "-c" statement.
  - -d Debug flag when starts the analyzer from a debugger. Prevents the system to kill the analyzer when the debugger stops at a breakpoint
  - -D Start analyzer as a daemon in the background (UNIX only).
  - -e <experiment> MIDAS experiment to connect to. (see odbedit task)
  - -f Filter mode. Write original events to output file only if the analyzer accepts them (doesn't return ANA_SKIP).
  - -h <hostname> MIDAS host to connect to when running the analyzer online (see odbedit task)
  - -i <filename1> <filename2> Input file name. May contain a '%05d' to be replaced by the run number. Up to ten input files can be specified in one "-i" statement.
  - -l If set, don't load histos from last histo file when running online.
  - -L HBOOK LREC size. Default is 8190.
  - -n <count> Analyze only "count" events.
  - -n <first> <last> Analyze only events from "first" to "last".
  - -n <first> <last> <n> Analyze every n-th event from "first" to "last".

- -o <filename> Output file name. Extension may be .mid (MIDAS binary), .asc (ASCII) or .rz (HBOOK). If the name contains a '%05d', one output file is generated for each run. Use "OFLN" as output file name to creaate a HBOOK shared memory instead of a file.
- -p <param=value> Set individual parameters to a specific value. Overrides any setting in configuration files
- -P <ODB tree> Protect an ODB subtree from being overwritten with the online data when ODB gets loaded from .mid file
- -q Quiet flag. If set, don't display run progress in offline mode.
- -r <range> Range of run numbers to analyzer like "-r 120 125" to analyze runs 120 to 125 (inclusive). The "-r" flag must be used with a '%05d' in the input file name.
- -s <port#> Specify the ROOT server TCP/IP port number (default 9090).
- -v Verbose output.
- -w Produce row-wise N-tuples in outpur .rz file. By default, column-wise N-tuples are used.

- **Remarks**
  - The creation of the **experim.h** is done through the **odbedit**> make <analyzer>. In order to include your **analyzer** section, the ODB /<**Analyzer**>/**Parameters** has to be present.

- **Usage**

```
>analyzer
>analyzer -D -r 9092
>analyzer -i run00023.mid -o run00023.rz -w
>analyzer -i run%05d.mid -o runall.rz -r 23 75 -w
```

6.15.7 mlogger task

mlogger is the main application to collect data from the different frontends under certain conditions and store them onto physical device such as *disk* or *tape*. It also acts as a history event collector if either the history flags are enabled in the frontend equipment (see The Equipment structure or if the ODB tree /History/Links is defined (See History system). See the ODB /Logger Tree for reference on the tree structure.

- **Arguments**
  - [-h] : help
  - [-e exptname] : experiment name (see odbedit task)

- [-D] : start program as a daemon (UNIX only).
- [-s] : Save mode (debugging: protect ODB).
- [-v] : Verbose (not to be used in conjunction with -D).

- **Usage**

```
>mlogger -D
```

- **Remarks**
  - The **mlogger** application requires to have an existing **/Equipment/** tree in the ODB!
  - As soon as the mlogger starts to run, the history mechanism is enabled.
  - The data channels as well as the history logging is rescanned automatically at each "begin of run" transition. In other word, additional channel can be defined while running but effect will take place only at the following begin of run transition.
  - The default setting defines a data "Midas" format with a file name of the type "run\%05d.mid". Make sure this is the requested setting for your experiment.
  - Once the mlogger is running, you should be able to monitor its state. through the mstat task or through the mhttpd task web browser.
  - From version 1.9.5
    - mlogger will not run if started remotely (argument -h hostname has been removed).
    - The file size limitation (<2GB) has been removed for older OS version.
    - mySQL data entry support.

6.15.8 lazylogger task

lazylogger is an application which decouples the data aquisition from the data logging mechanism. The need of such application has been dictated by the slow response time of some of the media logging devices (Tape devices). Delay due to tape mounting, retension, reposition implies that the data acquisition has to hold until operation completion. By using **mlogger** to log data to disk in a first stage and then using **lazylogger** to copy or move the stored files to the "slow device" we can keep the acquisition running without interruption.

- Multiple lazyloggers can be running comtemporary on the same computer, each one taking care of a particular channel.

- Each lazylogger channels will have a dedicated ODB tree containg its own information.
- All the lazylogger channel will be under the ODB **/Lazy/**<**channel_name**>/...
- Each channel tree is composed of three sub-tree **Settings**, Statistics, List.

Self-explanatory the **Settings** and Statistics contains the running operation of the channel. While the **List-** will have a dynamic list of run number which has been sucessfully manipulated by the Lazylogger channel. This list won't exist until the first successful operation of the channel is completed.

- **Arguments**
  - [-h] : help.
  - [-h hostname] : host name.
  - [-e exptname] : experiment name.
  - [-D] : start program as a daemon.
  - [-c channel] : logging channel. Specify the lazylogger to activate.
  - [-z] : zap statistics. Clear the statistics tree of all the defined lazylogger channels.

- **ODB parameters (Settings/)**

```
  Settings        DIR
      Maintain free space(%)    INT     1     4     3m   0   RWD  0
      Stay behind               INT     1     4     3m   0   RWD  -3
      Alarm Class               STRING  1     32    3m   0   RWD
      Running condition         STRING  1     128   3m   0   RWD  ALWAYS
      Data dir                  STRING  1     256   3m   0   RWD  /home/midas/online
      Data format               STRING  1     8     3m   0   RWD  MIDAS
      Filename format           STRING  1     128   3m   0   RWD  run%05d.mid
      Backup type               STRING  1     8     3m   0   RWD  Tape
      Execute after rewind      STRING  1     64    3m   0   RWD
      Path                      STRING  1     128   3m   0   RWD
      Capacity (Bytes)          FLOAT   1     4     3m   0   RWD  5e+09
      List label                STRING  1     128   3m   0   RWD
Execute before writing file     STRING  1     64    11h  0   RWD  lazy_prewrite.csh
Execute after writing file      STRING  1     64    11h  0   RWD  rundb_addrun.pl
Modulo.Position                 STRING  1     8     11h  0   RWD  2.1
Tape Data Append                BOOL    1     4     11h  0   RWD  y
```

  - **[Maintain free space]** As the Data Logger (mlogger) runs independently from the Lazylogger, the disk contains all the recorded data files. Under this condition, Lazylogger can be instructed to "purge" the data logging device (disk) after successful backup of the data onto the "slow device". The *Maintain* free space(%) parameter controls (if none zero) the percentage of disk space required to be maintain free.

* The condition for removing a data file is defined as:

  The data file corresponding to the given run number following the format declared under "Settings/Filename format" IS PRESENT on the "Settings/Data Dir" path. AND The given run number appears anywhere under the "List/" directory of ALL the Lazy channel having the same "Settings/Filename format"as this channel. AND The given run number appears anywhere under the "List/" directory of that channel

– **[Stay behind]** This parameter defines how many consecutive data files should be kept between the current run and the last lazylogger run.

  * **Example with "Stay behind = -3"** :

  1. Current acquisition run number 253 -> run00253.mid is being logged by mlogger.
  2. Files available on the disk corresponding to run #248, #249, #250, #251, #252.
  3. Lazylogger will start backing up run #250 as soon as the new run 254 starts. -3 "Stay behind = -3" corresponds to 3 file untouched on the disk (#251, #252, #253). The negative sign instructs the lazylogger to **always** scan the entire "Data Dir" from the oldest to the most recent file sitting on the disk at the "Data Dir" path- for backup. If the "Stay behind" is positive, lazylogger will **backup** starting from- x behind the current acquisition run number. Run order will be ignored.

– **[Alarm Class]** Specify the Alarm class to be used in case of triggered alarm.

– **[Running condition]** Specify the type of condition for which the lazylogger should be actived. By default lazylogger is **ALWAYS** running. In the case of high data rate acquisition it could be necessary to activate the lazylogger only when the run is either paused, stopped or when some external condition is satisfied such as "Low beam intensity". In this latter case, condition based on a single field of the ODB can be given to establish when the application should be active.

  * **Example** :

```
odbedit> set "Running condition" WHILE_ACQ_NOT_RUNNING
odbedit> set "Running condition" "/alias/max_rate \< 200"
```

– **[Data dir]** Specify the Data directory path of the data files. By default if the "/Logger/Data Dir" is present, the pointed value is taken otherwise the current directory where lazylogger has been started is used.

– **[Data format]** Specify the Data format of the data files. Currently supported formats are: **MIDAS** and **YBOS**.

– **[Filename format]** Specify the file format of the data files. Same format as given for the data logger.

– **[Backup type]** Specify the "slow device" backup type. Default **Tape**. Can be **Disk** or **Ftp**.

- **[Execute after rewind]** Specify a script to run after completion of a lazylogger backup set (see below "Capacity (Bytes)").
- **[Path]** Specify the "slow device" path. Three possible types of Path:
  - For Tape : **/dev/nst0-** (UNIX like).
  - For Disk : **/data1/myexpt**
  - For Ftp : **host**,port,user,password,directory
- **[Capacity (Bytes)]** Specify the maximum "slow device" capacity in bytes. When this capacity is reached,the lazylogger will close the backup device and clear the "List Label" field to prevent further backup (see below). It will aslo rewind the stream device if possible.
- **[List label]** Specify a label for a set of backed up files to the "slow device". This label is used only internaly by the lazylogger for creating under the "/List" a new array composed of the backed up runs until the "Capacity" value has been reached. As the backup set is complete, lazylogger will clear this field and therefore prevent any further backup until a none empty label list is entered again. In the other hand the list label will remain under the "/List" key to display all run being backed up until the corresponding files have been removed from the disk.
- **[Exec preW file]** Permits to run a script before the begining of the lazy job. The **arguments** passed to the scripts are: input file name , output file name, current block number.
- **[Exec postW file]** Permits to run a script after the completion of the lazy job. The **arguments** passed to the scripts are: list label, current job number, source path, file name, file size in MB, current block number.
- **[Modulo.Position]** This field is for multiple instances of the lazylogger where each instance works on a sub-set of run number. By specifying the **Modulo.Position** you're telling the current lazy instance how many instances are simultaneously running (3.) and the position of which this instance is assigned to (.1). As an example for 3 lazyloggers running contemporaneously the field assignment should be :

```
Channel    Field    Run#
Lazy_1     3.0      21, 24, 27, ...
Lazy_2     3.1      22, 25, 28, ...
Lazy_3     3.2      23, 26, 29, ...
```

- **[Tape Data Append]** Enable the spooling of the Tape device to the End_-of_Device (EOD) before starting the lazy job. This command is valid only for "Backup Type" Tape. If this flag is not enabled the lazy job starts at the current tape position.
- **[Statistics/]** ODB tree specifying general information about the status of the current lazylogger channel state.
- **[List/]** ODB tree, will contain arrays of run number associated with the array name backup-set label. Any run number appearing in any of the arrays is considered to have been backed up.

- **Usage** lazylogger requires to be setup prior data file can be moved. This setup consists of 4 steps:
  - **[Step 1]** Invoking the lazylogger once for setting up the appropriate ODB tree and exit.

```
>lazylogger -c Tape
```

  - **[Step 2]** Edit the newly created ODB tree. Correct the setting field to match your requirement.

```
> odbedit -e midas
[local:midas:Stopped]/>cd /Lazy/tape/
[local:midas:Stopped]tape>ls
[local:midas:Stopped]tape>ls -lr
Key name                        Type    #Val  Size  Last Opn Mode Value
---------------------------------------------------------------------------
tape                            DIR
    Settings                    DIR
        Maintain free space(%)  INT     1     4     3m   0   RWD  0
        Stay behind             INT     1     4     3m   0   RWD  -3
        Alarm Class             STRING  1     32    3m   0   RWD
        Running condition       STRING  1     128   3m   0   RWD  ALWAYS
        Data dir                STRING  1     256   3m   0   RWD  /home/midas/online
        Data format             STRING  1     8     3m   0   RWD  MIDAS
        Filename format         STRING  1     128   3m   0   RWD  run%05d.mid
        Backup type             STRING  1     8     3m   0   RWD  Tape
        Execute after rewind    STRING  1     64    3m   0   RWD
        Path                    STRING  1     128   3m   0   RWD
        Capacity (Bytes)        FLOAT   1     4     3m   0   RWD  5e+09
        List label              STRING  1     128   3m   0   RWD
    Statistics                  DIR
        Backup file             STRING  1     128   3m   0   RWD  none
        File size [Bytes]       FLOAT   1     4     3m   0   RWD  0
        KBytes copied           FLOAT   1     4     3m   0   RWD  0
        Total Bytes copied      FLOAT   1     4     3m   0   RWD  0
        Copy progress [%]       FLOAT   1     4     3m   0   RWD  0
        Copy Rate [bytes per s] FLOAT   1     4     3m   0   RWD  0
        Backup status [%]       FLOAT   1     4     3m   0   RWD  0
        Number of Files         INT     1     4     3m   0   RWD  0
        Current Lazy run        INT     1     4     3m   0   RWD  0
[local:midas:Stopped]tape>cd Settings/
[local:midas:Stopped]Settings>set "Data dir" /data
[local:midas:Stopped]Settings>set "Capacity (Bytes)" 15e9
```

  - **[Step 3]** Start lazylogger in the background

```
>lazylogger -c Tape -D
```

  - **[Step 4]** At this point the lazylogger is running and waiting for the "list label" to be defined before starting the copy procedure. mstat task will display information regarding the status of the lazylogger.

```
> odbedit -e midas
[local:midas:Stopped]/>cd /Lazy/tape/Settings
[local:midas:Stopped]Settings>set "List label" cni-043
```

- **Remarks**
  - For every major operation of the lazylogger a message is sent to the Message buffer and will be appended to the default Midas log file (midas.log). These messages are the only mean of finding out What/When/Where/How the lazylogger has operated on a data file. See below a fragment of the midas::log for the chaos experiment. In this case the **Maintain** free space() field was enabled which produces the cleanup of the data files and the entry in the **List** tree after copy.

```
Fri Mar 24 14:40:08 2000 [Lazy_Tape] 8351 (rm:16050ms) /scr0/spring2000/run08351.ybs file I
Fri Mar 24 14:40:08 2000 [Lazy_Tape] Tape run#8351 entry REMOVED
Fri Mar 24 14:59:55 2000 [Logger] stopping run after having received 1200000 events
Fri Mar 24 14:59:56 2000 [CHAOS] Run 8366 stopped
Fri Mar 24 14:59:56 2000 [Logger] Run #8366 stopped
Fri Mar 24 14:59:57 2000 [SUSIYBOS] saving info in run log
Fri Mar 24 15:00:07 2000 [Logger] starting new run
Fri Mar 24 15:00:07 2000 [CHAOS] Run 8367 started
Fri Mar 24 15:00:07 2000 [Logger] Run #8367 started
Fri Mar 24 15:06:59 2000 [Lazy_Tape] cni-043[15] (cp:410.6s) /dev/nst0/run08365.ybs 864.02(
Fri Mar 24 15:07:35 2000 [Lazy_Tape] 8352 (rm:25854ms) /scr0/spring2000/run08352.ybs file I
Fri Mar 24 15:07:35 2000 [Lazy_Tape] Tape run#8352 entry REMOVED
Fri Mar 24 15:27:09 2000 [Lazy_Tape] 8353 (rm:23693ms) /scr0/spring2000/run08353.ybs file I
Fri Mar 24 15:27:09 2000 [Lazy_Tape] Tape run#8353 entry REMOVED
Fri Mar 24 15:33:22 2000 [Logger] stopping run after having received 1200000 events
Fri Mar 24 15:33:22 2000 [CHAOS] Run 8367 stopped
Fri Mar 24 15:33:23 2000 [Logger] Run #8367 stopped
Fri Mar 24 15:33:24 2000 [SUSIYBOS] saving info in run log
Fri Mar 24 15:33:33 2000 [Logger] starting new run
Fri Mar 24 15:33:34 2000 [CHAOS] Run 8368 started
Fri Mar 24 15:33:34 2000 [Logger] Run #8368 started
Fri Mar 24 15:40:18 2000 [Lazy_Tape] cni-043[16] (cp:395.4s) /dev/nst0/run08366.ybs 857.67'
Fri Mar 24 15:50:15 2000 [Lazy_Tape] 8354 (rm:28867ms) /scr0/spring2000/run08354.ybs file I
Fri Mar 24 15:50:15 2000 [Lazy_Tape] Tape run#8354 entry REMOVED
...
```

  - Once lazylogger has started a job on a data file, trying to terminate the application will result in producing a log message informing the actual percentage of the backup completed so far. This message will repeat it self until completion of the backup and only then the lazylogger application will terminate.
  - If an interruption of the lazylogger is forced (kill...) The state of the backup device is undertermined. Recovery is not possible and the full backup set has to be redone. In order to do this, you need:
  - To rewind the backup device.
  - Delete the /Lazy/<channel_name>/List/<list label> array.
  - Restart the lazylogger with the -z switch which will "zap" the statistics entries.
  - In order to facilitate the recovery procedure, **lazylogger** produces an ODB ASCII file of the lazy channel tree after completion of successful operation.

This file (**Tape_recover.odb**) stored in Data_Dir can be used for ODB as well as lazylogger recovery.

6.15.9 mdump task

This application allows to "peep" into the data flow in order to display a snap-shot of the event. Its use is particularly powerful during experiment setup. In addition **mdump** has the capability to operate on data save-set files stored on disk or tape. The main **mdump** restriction is the fact that it works only for events formatted in banks (i.e.: MIDAS, YBOS bank).

- **Arguments** for Online
  - [-h] : help for online use.
  - [-h hostname] : Host name.
  - [-e exptname] : Experiment name.
  - [-b bank name] : Display event containg only specified bank name.
  - [-c compose] : Retrieve and compose file with either Add run# or Not (def:N).
  - [-f format] : Data representation (x/d/ascii) def:hex.
  - [-g type] : Sampling mode either Some or All (def:S). >>> in case of -c it is recommented to used -g all.
  - [-i id] : Event Id.
  - [-j] : Display bank header only.
  - [-k id] : Event mask. >>> -i and -k are valid for YBOS ONLY if EVID bank is present in the event
  - [-l number] : Number of consecutive event to display (def:1).
  - [-m mode] : Display mode either Bank or Raw (def:B)
  - [-p path] : Path for file composition (see -c)
  - [-s] : Data transfer rate diagnositic.
  - [-w time] : Insert wait in [sec] between each display.
  - [-x filename] : Input channel. data file name of data device. (def:online)
  - [-y] : Display consistency check only.
  - [-z buffer name] : Midas buffer name to attach to (def:SYSTEM)

- Additional **Arguments** for Offline
  - [-x -h] : help for offline use.

  - [-t type] : Bank format (Midas/Ybos). >>> if -x is a /dev/xxx, -t has to be specified.
  - [-r #] : skip record(YBOS) or event(MIDAS) to #.
  - [-w what] : Header, Record, Length, Event, Jbank_list (def:E) >>> Header & Record are not supported for MIDAS as it has no physical record structure.

- **Usage** mdump can operate on either data stream (online) or on save-set data file. Specific help is available for each mode.

```
 > mdump -h
 > mdump -x -h

Tue> mdump -x run37496.mid | more
------------------------ Event# 0 --------------------------------
------------------------ Event# 1 --------------------------------
Evid:0001- Mask:0100- Serial:1- Time:0x393c299a- Dsize:72/0x48
#banks:2 - Bank list:-SCLRRATE-

Bank:SCLR Length: 24(I*1)/6(I*4)/6(Type) Type:Integer*4
   1-> 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000

Bank:RATE Length: 24(I*1)/6(I*4)/6(Type) Type:Real*4 (FMT machine dependent)
   1-> 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
------------------------ Event# 2 --------------------------------
Evid:0001- Mask:0004- Serial:1- Time:0x393c299a- Dsize:56/0x38
#banks:2 - Bank list:-MMESMMOD-

Bank:MMES Length: 24(I*1)/6(I*4)/6(Type) Type:Real*4 (FMT machine dependent)
   1-> 0x3de35788 0x3d0b0e29 0x00000000 0x00000000 0x3f800000 0x00000000

Bank:MMOD Length: 4(I*1)/1(I*4)/1(Type) Type:Integer*4
   1-> 0x00000001
------------------------ Event# 3 --------------------------------
Evid:0001- Mask:0008- Serial:1- Time:0x393c299a- Dsize:48/0x30
#banks:1 - Bank list:-BMES-

Bank:BMES Length: 28(I*1)/7(I*4)/7(Type) Type:Real*4 (FMT machine dependent)
   1-> 0x443d7333 0x444cf333 0x44454000 0x4448e000 0x43bca667 0x43ce0000 0x43f98000
------------------------ Event# 4 --------------------------------
Evid:0001- Mask:0010- Serial:1- Time:0x393c299a- Dsize:168/0xa8
#banks:1 - Bank list:-CMES-

Bank:CMES Length: 148(I*1)/37(I*4)/37(Type) Type:Real*4 (FMT machine dependent)
   1-> 0x3f2f9fe2 0x3ff77fd6 0x3f173fe6 0x3daeffe2 0x410f83e8 0x40ac07e3 0x3f6ebfd8 0x3c47ffde
   9-> 0x3e60ffda 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x3f800000
  17-> 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
  25-> 0x3f800000 0x3f800000 0x3f800000 0x00000000 0x3f800000 0x00000000 0x3f800000 0x3f800000
  33-> 0x3f800000 0x3f800000 0x3f800000 0x3f800000 0x00000000
------------------------ Event# 5 --------------------------------
Evid:0001- Mask:0020- Serial:1- Time:0x393c299a- Dsize:32/0x20
#banks:1 - Bank list:-METR-

Bank:METR Length: 12(I*1)/3(I*4)/3(Type) Type:Real*4 (FMT machine dependent)
```

```
  1-> 0x00000000 0x39005d87 0x00000000
...
```

- **Example**

```
> mdump -j
```

6.15.10 mevb task

mevb is an event builder application taking several frontends Midas data source and assembles a new overall Midas event.

In the case where overall data collection is handled by multiple physically separated frontends, it could be necessary to assemble these data fragments into a dedicated event. The synchonization of the fragment collection is left to the user which is done usually through specific hardware mechanism. Once the fragments are composed in each frontend, they are sent to the "Event Builder" (eb) where the serial number (pheader->serial_number) of each fragment is compared one event at a time for serial match. In case of match, a new event will be composed with its own event ID and serial number followed by all the expected fragments. The composed event is then sent to the next stage which is usually the data logger (mlogger).

The mhttpd task will present the status of the event builder as an extra equipment with its corresponding statistical information.

- **Arguments**
  - [-h] : help
  - [-h hostname] : host name
  - [-e exptname] : experiment name
  - [-b] : Buffer name
  - [-v] : Show wheel
  - [-d] : debug messages
  - [-D] : start program as a daemon
- **Usage**

```
Thu> mevb -e midas
Program mevb/EBuilder version 2 started
```

- See Event Builder Functions for more details

6.15.11 mspeaker, mlxspeaker tasks

mspeaker, mlxspeaker are utilities which listen to the Midas messages system and pipe these messages to a speech synthesizer application. **mspeaker** is for the Windows based system and interface to the `FirstByte/ProVoice package`. The **mlxspeaker** is for Linux based system and interface to the `Festival`. In case of use of either package, the speech synthesis system has to be installed prior to the activation of the **mspeaker**, mlxspeaker.

- **Arguments**
  - [-h] : help
  - [-h hostname] : host name
  - [-e exptname] : experiment name
  - [-t mt_talk_cmd] : Specify the talk alert command (ux only).
  - [-u mt_user_cmd] : Specify the user alert command (ux only).
  - [-s shut up time]: Specify the min time interval between alert [s] The -t & -u switch require a command equivalent to: '-t play –volume=0.3 file.wav'
  - [-D] : start program as a daemon

- **Usage**

```
> mlxspeaker -D
```

6.15.12 mcnaf task

mcnaf is an interactive CAMAC tool which allows "direct" access to the CAMAC hardware. This application is operational under either of the two following conditions:

1. **mcnaf** has been built against a particular CAMAC driver (see CAMAC drivers).
2. A user frontend code using a valid CAMAC driver is currently active. In this case the frontend acts as a RPC CAMAC server and will handle the CAMAC request. This last option is only available if the frontend code (mfe.c) from the Building Options has included the HAVE_CAMAC pre-compiler flag.

- **Arguments**
  - [-h] : help
  - [-h hostname] : host name

  - [-e exptname] : experiment name
  - [-f frontend name] : Frontend name to connect to.
  - [-s RPC server name] : CAMAC RPC server name for remote connection.

- **Building application** The **midas/utils/makefile**.mcnaf will build a collection of **mcnaf** applications which are hardware dependent, see **Example** below:
  - **[miocnaf]** cnaf application using the declared CAMAC hardware DRIVER (kcs2927 in this case). To be used with **dio** CAMAC application starter (see dio task).
  - **[mwecnaf]** cnaf application using the WI-E-N-ER PCI/CAMAC interface (see CAMAC drivers). Please contact: midas@triumf.ca for further information.
  - **[mcnaf]** cnaf application using the CAMAC RPC capability of any Midas frontend program having CAMAC access.
  - **[mdrvcnaf]** cnaf application using the Linux CAMAC driver for either kcs2927, kcs2926, dsp004. This application would require to have the proper Linux module loaded in the system first. Please contact mailto:midas@triumf.ca for further information.

```
Thu> cd /midas/utils
Thu> make -f makefile.mcnaf DRIVER=kcs2927
gcc -O3 -I../include -DOS_LINUX -c -o mcnaf.o mcnaf.c
gcc -O3 -I../include -DOS_LINUX -c -o kcs2927.o ../drivers/bus/kcs2927.c
gcc -O3 -I../include -DOS_LINUX -o miocnaf mcnaf.o kcs2927.o  ../linux/lib/libmidas.a -lutil
gcc -O3 -I../include -DOS_LINUX -c -o wecc32.o ../drivers/bus/wecc32.c
gcc -O3 -I../include -DOS_LINUX -o mwecnaf mcnaf.o wecc32.o  ../linux/lib/libmidas.a -lutil
gcc -O3 -I../include -DOS_LINUX -c -o camacrpc.o ../drivers/bus/camacrpc.c
gcc -O3 -I../include -DOS_LINUX -o mcnaf mcnaf.o camacrpc.o  ../linux/lib/libmidas.a -lutil
gcc -O3 -I../include -DOS_LINUX -c -o camaclx.o ../drivers/bus/camaclx.c
gcc -O3 -I../include -DOS_LINUX -o mdrvcnaf mcnaf.o camaclx.o  ../linux/lib/libmidas.a -lutil
rm *.o
```

- **Running application**
  - Direct CAMAC access: This requires the computer to have the proper CAMAC interface installed and the **BASE** ADDRESS matching the value defined in the corresponding CAMAC driver. For kcs2926.c, kcs2927.c, dsp004.c, hyt1331.c, the base address (CAMAC_BASE) is set to 0x280.

    ```
    >dio miocnaf
    ```

  - RPC CAMAC through frontend: This requires to have a frontend running which will be able to serve the CAMAC RPC request. Any Midas frontend has that capability built-in but it has to have the proper CAMAC driver included in it.

    ```
    >mcnaf -e <expt> -h <host> -f <fe_name>
    ```

- **Usage**

  . . . . . .

6.15.13 melog task

Electronic Log utility. Submit full Elog entry to the specified Elog port.

- **Arguments**
  - [-h] : help
  - [-h hostname] : host name
  - [-l exptname or logbook]
  - [-u username password]
  - [-f <attachment>] : up to 10 files.
  - -a <attribute>=<value> : up to 20 attributes. The attribute "Author=..." must at least be present for submission of Elog.
  - -m <textfile> | text> Arguments with blanks must be enclosed in quotes. The elog message can either be submitted on the command line or in a file with the -m flag. Multiple attributes and attachments can be supplied.

- **Usage** By default the attributes are "Author", "Type", "System" and "Subject". The "Author" attribute has to be present in the elog command in order to successfully submit the message. If multiple attributes are required append before "text" field the full specification of the attribute. In case of multiple attachements, only one "-f" is required followed by up to 10 file names.

```
>melog -h myhost -p 8081 -l myexpt -a author=pierre "Just a elog message"
>melog -h myhost -p 8081 -l myexpt -a author=pierre -f file2attach.txt \
                "Just this message with an attachement"
>melog -h myhost -p 8081 -l myexpt -a author=pierre -m file_containing_the_message.txt
>melog -h myhost -p 8081 -l myexpt -a Author=pierre -a Type=routine -a system=general \
                -a Subject="my test" "A full Elog message"
```

- **Remarks**

6.15.14 mhist task

History data retriever.

- **Arguments**
  - [-h] : help
  - [-e Event ID] : specify event ID
  - [-v Variable Name] : specify variable name for given Event ID
  - [-i Index] : index of variables which are arrays
  - [-i Index1:Index2] index range of variables which are arrays (max 50)
  - [-t Interval] : minimum interval in sec. between two displayed records
  - [-h Hours] : display between some hours ago and now
  - [-d Days] : display between some days ago and now
  - [-f File] : specify history file explicitly
  - [-s Start date] : specify start date DDMMYY[.HHMM[SS]]
  - [-p End date] : specify end date DDMMYY[.HHMM[SS]]
  - [-l] : list available events and variables
  - [-b] : display time stamp in decimal format
  - [-z] : History directory (def: cwd).
- **Usage**
- **Example**

```
--- All variables of event ID 9 during last hour with at least 5 minutes interval.
> mhist
Available events:
ID 9: Target
ID 5: CHV
ID 6: B12Y
ID 20: System

Select event ID: 9

Available variables:
0: Time
1: Cryostat vacuum
2: Heat Pipe pressure
3: Target pressure
4: Target temperature
5: Shield temperature
6: Diode temperature

Select variable (0..6,-1 for all): -1
```

```
How many hours: 1

Interval [sec]: 300

Date    Time    Cryostat vacuum Heat Pipe pressure  Target pressure Target temperature      S
Jun 19 10:26:23 2000    104444  4.614   23.16   -0.498  22.931  82.163  40
Jun 19 10:31:24 2000    104956  4.602   23.16   -0.498  22.892  82.108  40
Jun 19 10:36:24 2000    105509  4.597   23.099  -0.498  22.892  82.126  40
Jun 19 10:41:33 2000    110021  4.592   23.16   -0.498  22.856  82.08   40
Jun 19 10:46:40 2000    110534  4.597   23.147  -0.498  22.892  82.117  40
Jun 19 10:51:44 2000    111046  4.622   23.172  -0.498  22.907  82.117  40
Jun 19 10:56:47 2000    111558  4.617   23.086  -0.498  22.892  82.117  40
Jun 19 11:01:56 2000    112009  4.624   23.208  -0.498  22.892  82.117  40
Jun 19 11:07:00 2000    112521  4.629   23.172  -0.498  22.896  82.099  40
Jun 19 11:12:05 2000    113034  4.639   23.074  -0.498  22.896  82.117  40
Jun 19 11:17:09 2000    113546  4.644   23.172  -0.498  22.892  82.126  40
Jun 19 11:22:15 2000    114059  4.661   23.16   -0.498  22.888  82.099  40
```

- Single variable "I-WC1+_Anode" of event 5 every hour over the full April 24/2000.

```
 mhist -e 5 -v "I-WC1+_Anode" -t 3600 -s 240400 -p 250400
Apr 24 00:00:09 2000    160
Apr 24 01:00:12 2000    160
Apr 24 02:00:13 2000    160
Apr 24 03:00:14 2000    160
Apr 24 04:00:21 2000    180
Apr 24 05:00:26 2000    0
Apr 24 06:00:31 2000    160
Apr 24 07:00:37 2000    160
Apr 24 08:00:40 2000    160
Apr 24 09:00:49 2000    160
Apr 24 10:00:52 2000    160
Apr 24 11:01:01 2000    160
Apr 24 12:01:03 2000    160
Apr 24 13:01:03 2000    0
Apr 24 14:01:04 2000    0
Apr 24 15:01:05 2000    -20
Apr 24 16:01:11 2000    0
Apr 24 17:01:14 2000    0
Apr 24 18:01:19 2000    -20
Apr 24 19:01:19 2000    0
Apr 24 20:01:21 2000    0
Apr 24 21:01:23 2000    0
Apr 24 22:01:32 2000    0
Apr 24 23:01:39 2000    0
```

- **Remarks** : History data can be retrieved and displayed through the Midas web page (see mhttpd task).

- **Example**

Midas Web History display.

Figure 38: Midas Web History display.

6.15.15 mchart task

mchart is a periodic data retriever of a specific path in the ODB which can be used in conjunction with a stripchart graphic program.

- In the first of two step procedure, a specific path in the ODB can be scanned for composing a configuration file by extracting all numerical data references **file.conf** .
- In the second step the mchart will produce at fix time interval a refreshed data file containing the values of the numerical data specified in the configuration file. This file is then available for a stripchart program to be used for chart recording type of graph.

Two possible stripchart available are:

- **gstripchart** The configuration file generated by mchart is compatible with the GNU stripchart which permits sofisticated data equation manipulation. On the other hand, the data display is not very fency and provides just a basic chart recorder.
- stripchart.tcl file This tcl/tk application written by Gertjan Hofman provides a far better graphical chart recorder display tool, it also permits history save-set display, but the equation scheme is not implemented.
- **Arguments**
  - [-h] : help
  - [-h hostname] : host name.
  - [-e exptname] : experiment name.
  - [-D] : start program as a daemon.
  - [-u time] : data update periodicity (def:5s).
  - [-f file] : file name (+.conf: if using existing file).
  - [-q ODBpath] : ODB tree path for extraction of the variables.
  - [-c] : ONLY creates the configuration file for later use.
  - [-b lower_value] : sets general lower limit for all variables.
  - [-t upper_value] : sets general upper limit for all variables.
  - [-g] : spawn the graphical stripchart if available.
  - [-gg] : force the use of gstripchart for graphic.
  - [-gh] : force the use of stripchart (tcl/tk) for graphic.
- **Usage** : The configuration contains one entry for each variable found under the ODBpath requested. The format is described in the gstripchart documentation.

Once the configuration file has been created, it is possible to apply any valid operation (equation) to the parameters of the file following the gstripchart syntax.

In the case of the use of the *stripchart* from G.Hofman, only the "filename", "pattern", "maximum", "minimum" fields are used.

When using mchart with -D Argument, it is necessary to have the MCHART_DIR defined in order to allow the daemon to find the location of the configuration and data files (see Environment variables).

```
chaos:~/chart> more trigger.conf
#Equipment:            >/equipment/kos_trigger/statistics
menu:                   on
slider:                 on
type:                   gtk
minor_ticks:            12
```

```
major_ticks:            6
chart-interval:         1.000
chart-filter:           0.500
slider-interval:        0.200
slider-filter:          0.200
begin:        Events_sent
  filename:     /home/chaos/chart/trigger
  fields:       2
  pattern:      Events_sent
  equation:     \$2
  color:        \$blue
  maximum:      1083540.00
  minimum:      270885.00
  id_char:      1
end:            Events_sent
begin:        Events_per_sec.
  filename:     /home/chaos/chart/trigger
  fields:       2
  pattern:      Events_per_sec.
  equation:     $2
  color:        \$red
  maximum:      1305.56
  minimum:      326.39
  id_char:      1
end:            Events_per_sec.
begin:        kBytes_per_sec.
  filename:     /home/chaos/chart/trigger
  fields:       2
  pattern:      kBytes_per_sec.
  equation:     $2
  color:        \$brown
  maximum:      898.46
  minimum:      224.61
  id_char:      1
end:            kBytes_per_sec.
```

A second file (data file) will be updated a fixed interval by the {*mchart}* utility.

```
chaos:~/chart> more trigger
  Events_sent 6.620470e+05
  Events_per_sec. 6.463608e+02
  kBytes_per_sec. 4.424778e+02
```

- **Example**

- Creation with ODBpath being one array and one element of 2 sitting under variables/:

```
chaos:~/chart> mchart -f chvv -q /equipment/chv/variables/chvv -c
chaos:~/chart> ls -l chvv*
-rw-r--r--   1 chaos    users         474 Apr 18 14:37 chvv
-rw-r--r--   1 chaos    users        4656 Apr 18 14:37 chvv.conf
```

- Creation with ODBpath of all the sub-keys sittings in variables:

```
mchart -e myexpt -h myhost -f chv -q /equipment/chv/variables -c
```

- Creation and running in debug:

```
chaos:~/chart> mchart -f chv -q /equipment/chv/variables -d
CHVV : size:68
#name:17 #Values:17
CHVI : size:68
```

- Running a pre-existing conf file (chv.conf) debug:

```
chaos:~/chart> mchart -f chv.conf -d
CHVV : size:68
#name:17 #Values:17
CHVI : size:68
#name:17 #Values:17
```

- Running a pre-existing configuration file and spawning gstripchart:

```
chaos:~/chart> mchart -f chv.conf -gg
spawning graph with gstripchart -g 500x200-200-800 -f /home/chaos/chart/chv.conf ...
```

- Running a pre-existing configuration file and spawning stripchart, this will work only if Tcl/Tk and bltwish packages are installed and the stripchart.tcl has been installed through the Midas Makefile.

```
chaos:~/chart> mchart -f chv.conf -gh
spawning graph with stripchart /home/chaos/chart/chv.conf ...
```

6.15.16 mtape task

Tape manipulation utility.

- **Arguments**
  - [-h] : help
  - [-h hostname] : host name
  - [-e exptname] : experiment name
  - [-D] : start program as a daemon
- **Usage**
- **Example**

```
>mtape
```

6.15.17 dio task

Direct I/O task provider (LINUX).

If no particular Linux driver is installed for the CAMAC access, the **dio-** program will allow you to access the I/O ports to which the CAMAC interface card is connected to.

- **Arguments**
  - [application name] : Program name requiring I/O permission.
- **Usage**

```
>dio miocnaf
>dio frontend
```

- **Remark**
- This "hacking" utility restricts the access to a range of I/O ports from 0x200 to 0x3FF.
- As this mode if I/O access by-passes the driver (if any), concurrent access to the same I/O port may produce unexpected result and in the worst case it will freeze the computer. It is therefore important to ensure to run one and only one dio application to a given port in order to prevent potential hangup problem.
- Interrupt handling, DMA capabilities of the interface will not be accessible under this mode of operation.

6.15.18 stripchart.tcl file

Graphical stripchart data display. Operates on mchart task data or on Midas history save-set files. (see also History system).

- **Arguments**
  - [-mhist] : start stripchart for Midas history data.
- **Usage** : stripchart <-options> <config-file> -mhist: (look at history file -default) -dmhist: debug mhist -debug: debug stripchart -config_file: see mchart_task

```
> stripchart.tcl -debug
> stripchart.tcl
```

- **Example**

```
> stripchart.tcl -h
```

gstripchart display with parameters and data pop-up.

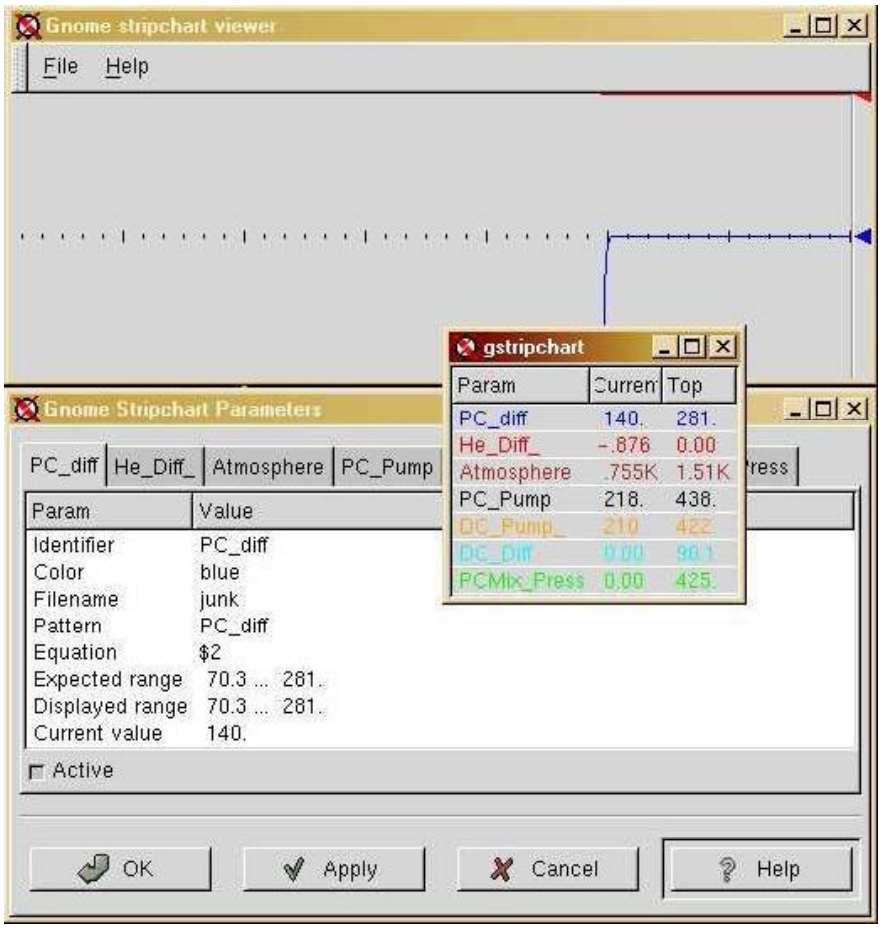


Figure 39: gstripchart display with parameters and data pop-up.

stripchart.tcl mhist mode: main window with pull-downs.

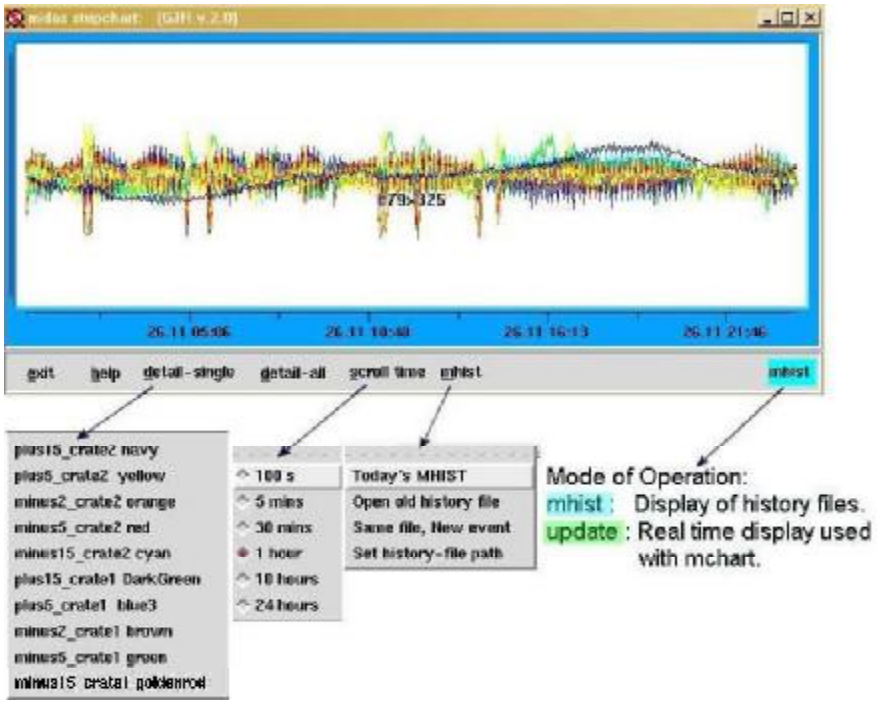


Figure 40: stripchart.tcl mhist mode: main window with pull-downs.

stripchart.tcl Online data, running in conjunction with mchart

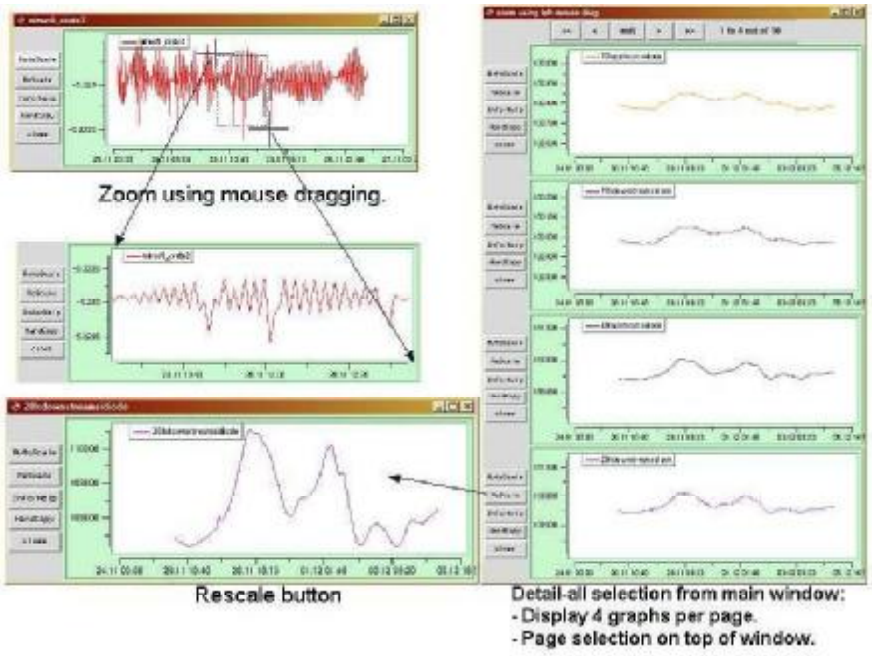


Figure 41: stripchart.tcl Online data, running in conjunction with mchart

6.15.19 rmidas task

Root/Midas remote GUI application for root histograms and possible run control under the ROOT. environment.

- **Arguments**
  - [-h] : help
  - [-h hostname] : host name
  - [-e exptname] : experiment name
- **Usage** to be written.
- **Example**

```
>rmidas midasserver.domain
```

rmidas display sample. Using the example/experiment/ demo setup.

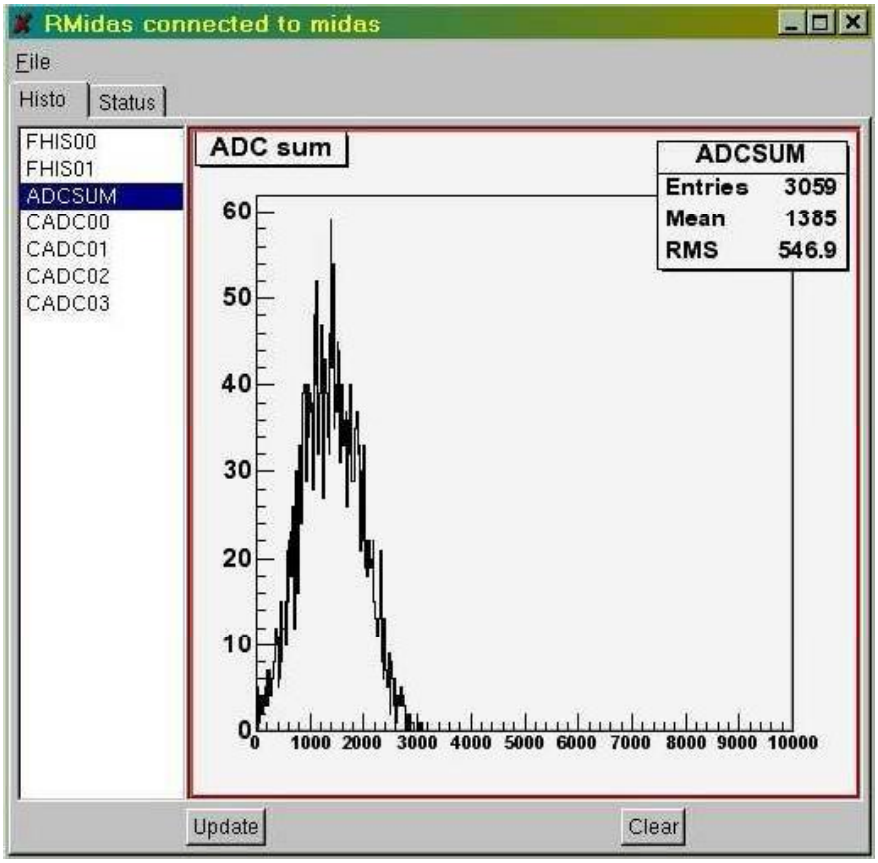


Figure 42: rmidas display sample. Using the example/experiment/ demo setup.

6.15.20 hvedit task

High Voltage editor, graphical interface to the Slow Control System. Originally for Windows machines, but recently ported on Linux under Qt by Andreas Suter.

- **Arguments**
  - [-h] : help
  - [-h hostname] : host name

  - [-e exptname] : experiment name
  - [-D] : start program as a daemon

- **Usage** : To control the high voltage system, the program HVEdit can be used under Windows 95/NT. It can be used to set channels, save and load values from disk and print them. The program can be started several times even on different computers. Since they are all linked to the same ODB arrays, the demand and measured values are consistent among them at any time. HVEdit is started from the command line:

- **Example**

```
>hvedit
```

6.15.21 Midas Remote server

mserver provides remote access to any midas client. This task usually runs in the background and doesn't need to be modified. In the case where debugging is required, the *mserver* can be started with the -d flag which will write an entry for each transaction appearing onto the mserver. This log entry contains the time stamp and RPC call request.

- **Arguments**
  - [-h] : help
  - [-s] : Single process server
  - [-t] : Multi thread server
  - [-m] : Milti process server (default)
  - [-d] : Write debug info to /tmp/mserver.log
  - [-D] : Become a Daemon

- **Usage**