Midas Reference Manual

2.0.0-1

Generated by Doxygen 1.3.9.1

Fri Nov 2 10:31:07 2007

Contents

1 Midas Data Acquisition

Welcome to the world of Midas.

The System

Midas is a versatile Data acquisition System for medium scale physics experiments. This document will try to answer most of your questions regarding installation, setup, running, and development.

If you're looking for a flexible and simple DAQ, you may want to consider Midas and its applications. Feel free to browse through the following links.

- **Midas** is a result of a development effort made by Stefan Ritt at `PSI`/Switzerland (`Midas@PSI`) in collaboration with the members of the Data acquisition group at `Triumf`/Canada.

- Midas has a dedicated discussion forum (Electronic logbook) which provides common place for midas users to report problems, share experience or post improvement wishes. You can browse this `Elog` or register for Email notifications.

- The Midas source code is subject to the `GPL` and can be accessed from the `SVN` repository site in Switzerland either for inspection or download.

- **Tarball** are available either from the `Switzerland` or `Canadian` sites.

- The main purpose of the MIDAS DAQ is to provide:

  - data collection from local and/or remote hardware source.
  - data recording to common storage media.
  - Full data flow control.

- It does also event-by-event analysis through PAW or Root based application. Please refer to:
  - `ROME` analyzer framework.
  - `Roody` GUI histogram visualizer application.

- The current hardware support is listed at this `location`.

- Other related links can be found `here`.

- The following chapters refer to the online Midas documentation.

1.1 Content

- New Documented Features : Whats new in Midas.
- Introduction : Some initial words and description
- Components : Listing
- Quick Start : The HowTo for installation.
- Internal features : The main internal components of the system.
- MIDAS Analyzer : PAW/Root Analyzer.
- Utilities : The Midas applications.
- Data format : Supported data format
- Supported hardware : Supported hardware.
- CAMAC and VME access function call : CAMAC and VME access function call.
- Midas build options and operation considerations : Midas build options and operation consideration.
- Midas Code and Libraries : Midas Library.
- Frequently Asked Questions : Frequently Asked Questions.

2 Midas Module Documentation

2.1 Midas VME standard

Modules

- group VME Functions (mvme_xxx)

Data Structures

- struct MVME_INTERFACE

Defines

- #define MVME_SUCCESS 1
- #define MVME_DMODE_D8 1
- #define MVME_DMODE_D16 2
- #define MVME_DMODE_D32 3
- #define MVME_DMODE_D64 4
- #define MVME_DMODE_RAMD16 5
- #define MVME_DMODE_RAMD32 6
- #define MVME_DMODE_LM 7
- #define MVME_BLT_NONE 1
- #define MVME_BLT_BLT32 2
- #define MVME_BLT_MBLT64 3
- #define MVME_BLT_2EVME 4
- #define MVME_BLT_2ESST 5
- #define MVME_BLT_BLT32FIFO 6
- #define MVME_BLT_MBLT64FIFO 7
- #define MVME_BLT_2EVMEFIFO 8
- #define MVME_AM_A32_SB (0x0F)
- #define MVME_AM_A32_SP (0x0E)
- #define MVME_AM_A32_SD (0x0D)
- #define MVME_AM_A32_NB (0x0B)
- #define MVME_AM_A32_NP (0x0A)
- #define MVME_AM_A32_ND (0x09)
- #define MVME_AM_A32_SMBLT (0x0C)
- #define MVME_AM_A32_NMBLT (0x08)
- #define MVME_AM_A24_SB (0x3F)
- #define MVME_AM_A24_SP (0x3E)

- #define MVME_AM_A24_SD (0x3D)
- #define MVME_AM_A24_NB (0x3B)
- #define MVME_AM_A24_NP (0x3A)
- #define MVME_AM_A24_ND (0x39)
- #define MVME_AM_A24_SMBLT (0x3C)
- #define MVME_AM_A24_NMBLT (0x38)
- #define MVME_AM_A16_SD (0x2D)
- #define MVME_AM_A16_ND (0x29)

2.1.1 Define Documentation

2.1.1.1 #define MVME_ACCESS_ERROR 7

Definition at line 67 of file mvmestd.h.

2.1.1.2 #define MVME_AM_A16 MVME_AM_A16_SD

Definition at line 130 of file mvmestd.h.

2.1.1.3 #define MVME_AM_A16_ND (0x29)

A16 Short Non-Privileged Data Access

Definition at line 128 of file mvmestd.h.

2.1.1.4 #define MVME_AM_A16_SD (0x2D)

A16 Short Supervisory Data Access

Definition at line 127 of file mvmestd.h.

2.1.1.5 #define MVME_AM_A24 MVME_AM_A24_SD

Definition at line 124 of file mvmestd.h.

2.1.1.6 #define MVME_AM_A24_D64 MVME_AM_A24_SMBLT

Definition at line 125 of file mvmestd.h.

2.1.1.7 #define MVME_AM_A24_NB (0x3B)

A24 Standard Non-Privileged Block Transfer

Definition at line 118 of file mvmestd.h.

2.1.1.8 #define MVME_AM_A24_ND (0x39)

A24 Standard Non-Privileged Data Access

Definition at line 120 of file mvmestd.h.

2.1.1.9 #define MVME_AM_A24_NMBLT (0x38)

A24 Multiplexed Block Transfer (D64)

Definition at line 122 of file mvmestd.h.

2.1.1.10 #define MVME_AM_A24_NP (0x3A)

A24 Standard Non-Privileged Program Access

Definition at line 119 of file mvmestd.h.

2.1.1.11 #define MVME_AM_A24_SB (0x3F)

A24 Standard Supervisory Block Transfer

Definition at line 115 of file mvmestd.h.

2.1.1.12 #define MVME_AM_A24_SD (0x3D)

A24 Standard Supervisory Data Access

Definition at line 117 of file mvmestd.h.

2.1.1.13 #define MVME_AM_A24_SMBLT (0x3C)

A24 Multiplexed Block Transfer (D64)

Definition at line 121 of file mvmestd.h.

2.1.1.14 #define MVME_AM_A24_SP (0x3E)

A24 Standard Supervisory Program Access

Definition at line 116 of file mvmestd.h.

2.1.1.15 #define MVME_AM_A32 MVME_AM_A32_SD

Definition at line 112 of file mvmestd.h.

2.1.1.16 #define MVME_AM_A32_D64 MVME_AM_A32_SMBLT

Definition at line 113 of file mvmestd.h.

2.1.1.17 #define MVME_AM_A32_NB (0x0B)

A32 Extended Non-Privileged Block

Definition at line 106 of file mvmestd.h.

2.1.1.18 #define MVME_AM_A32_ND (0x09)

A32 Extended Non-Privileged Data

Definition at line 108 of file mvmestd.h.

2.1.1.19 #define MVME_AM_A32_NMBLT (0x08)

A32 Multiplexed Block Transfer (D64)

Definition at line 110 of file mvmestd.h.

2.1.1.20 #define MVME_AM_A32_NP (0x0A)

A32 Extended Non-Privileged Program

Definition at line 107 of file mvmestd.h.

2.1.1.21 #define MVME_AM_A32_SB (0x0F)

A32 Extended Supervisory Block

Definition at line 103 of file mvmestd.h.

2.1.1.22 #define MVME_AM_A32_SD (0x0D)

A32 Extended Supervisory Data

Definition at line 105 of file mvmestd.h.

2.1.1.23 #define MVME_AM_A32_SMBLT (0x0C)

A32 Multiplexed Block Transfer (D64)

Definition at line 109 of file mvmestd.h.

2.1.1.24 #define MVME_AM_A32_SP (0x0E)

A32 Extended Supervisory Program

Definition at line 104 of file mvmestd.h.

2.1.1.25 #define MVME_AM_DEFAULT MVME_AM_A32

Definition at line 132 of file mvmestd.h.

2.1.1.26 #define MVME_BLT_2ESST 5

two edge source synchrnous transfer

Definition at line 96 of file mvmestd.h.

2.1.1.27 #define MVME_BLT_2EVME 4

two edge block transfer

Definition at line 95 of file mvmestd.h.

2.1.1.28 #define MVME_BLT_2EVMEFIFO 8

two edge block transfer with FIFO mode

Definition at line 99 of file mvmestd.h.

2.1.1.29 #define MVME_BLT_BLT32 2

32-bit block transfer

Definition at line 93 of file mvmestd.h.

2.1.1.30 #define MVME_BLT_BLT32FIFO 6

FIFO mode, don't increment address

Definition at line 97 of file mvmestd.h.

2.1.1.31 #define MVME_BLT_MBLT64 3

multiplexed 64-bit block transfer

Definition at line 94 of file mvmestd.h.

2.1.1.32 #define MVME_BLT_MBLT64FIFO 7

FIFO mode, don't increment address

Definition at line 98 of file mvmestd.h.

2.1.1.33 #define MVME_BLT_NONE 1

normal programmed IO

Definition at line 92 of file mvmestd.h.

2.1.1.34 #define MVME_DMODE_D16 2

D16

Definition at line 81 of file mvmestd.h.

2.1.1.35 #define MVME_DMODE_D32 3

D32

Definition at line 82 of file mvmestd.h.

2.1.1.36 #define MVME_DMODE_D64 4

D64

Definition at line 83 of file mvmestd.h.

2.1.1.37 #define MVME_DMODE_D8 1

D8

Definition at line 80 of file mvmestd.h.

2.1.1.38 #define MVME_DMODE_DEFAULT MVME_DMODE_D32

Definition at line 88 of file mvmestd.h.

2.1.1.39 #define MVME_DMODE_LM 7

local memory mapped to VME

Definition at line 86 of file mvmestd.h.

2.1.1.40 #define MVME_DMODE_RAMD16 5

RAM memory of VME adapter

Definition at line 84 of file mvmestd.h.

2.1.1.41 #define MVME_DMODE_RAMD32 6

RAM memory of VME adapter

Definition at line 85 of file mvmestd.h.

2.1.1.42 #define MVME_INVALID_PARAM 5

Definition at line 65 of file mvmestd.h.

2.1.1.43 #define MVME_NO_CRATE 3

Definition at line 63 of file mvmestd.h.

2.1.1.44 #define MVME_NO_INTERFACE 2

Definition at line 62 of file mvmestd.h.

2.1.1.45 #define MVME_NO_MEM 6

Definition at line 66 of file mvmestd.h.

2.1.1.46 #define MVME_SUCCESS 1

dox***

Definition at line 61 of file mvmestd.h.

2.1.1.47 #define MVME_UNSUPPORTED 4

Definition at line 64 of file mvmestd.h.

2.1.2 Typedef Documentation

2.1.2.1 typedef unsigned int mvme_addr_t

Definition at line 71 of file mvmestd.h.

2.1.2.2 typedef unsigned int mvme_locaddr_t

Definition at line 72 of file mvmestd.h.

2.1.2.3 typedef unsigned int mvme_size_t

Definition at line 73 of file mvmestd.h.

2.2 VME Functions (mvme_xxx)

Functions

- int EXPRT mvme_open (MVME_INTERFACE ∗∗vme, int idx)
- int EXPRT mvme_close (MVME_INTERFACE ∗vme)
- int EXPRT mvme_sysreset (MVME_INTERFACE ∗vme)
- int EXPRT mvme_read (MVME_INTERFACE ∗vme, void ∗dst, mvme_addr_t vme_addr, mvme_size_t n_bytes)
- unsigned int EXPRT mvme_read_value (MVME_INTERFACE ∗vme, mvme_addr_t vme_addr)
- int EXPRT mvme_write (MVME_INTERFACE ∗vme, mvme_addr_t vme_addr, void ∗src, mvme_size_t n_bytes)
- int EXPRT mvme_write_value (MVME_INTERFACE ∗vme, mvme_addr_t vme_addr, unsigned int value)
- int EXPRT mvme_set_am (MVME_INTERFACE ∗vme, int am)
- int EXPRT mvme_get_am (MVME_INTERFACE ∗vme, int ∗am)
- int EXPRT mvme_set_dmode (MVME_INTERFACE ∗vme, int dmode)
- int EXPRT mvme_get_dmode (MVME_INTERFACE ∗vme, int ∗dmode)
- int EXPRT mvme_set_blt (MVME_INTERFACE ∗vme, int mode)
- int EXPRT mvme_get_blt (MVME_INTERFACE ∗vme, int ∗mode)

2.2.1 Function Documentation

2.2.1.1 int EXPRT mvme_close (MVME_INTERFACE ∗ *vme*)

Close and release ALL the opened VME channel. See example in mvme_open()

Parameters:

∗***vme*** VME structure.

Returns:

MVME_SUCCESS, MVME_ACCESS_ERROR

2.2.1.2 int EXPRT mvme_get_am (MVME_INTERFACE ∗ *vme*, int ∗ *am*)

Get Address Modifier.

Parameters:

∗***vme*** VME structure

∗***am*** returned address modifier

Returns:

MVME_SUCCESS

2.2.1.3 int EXPRT mvme_get_blt (MVME_INTERFACE ∗ *vme*, int ∗ *mode*)

Get current Data mode.

Parameters:

∗***vme*** VME structure

∗***mode*** returned BLT mode

Returns:

MVME_SUCCESS

2.2.1.4 int EXPRT mvme_get_dmode (MVME_INTERFACE ∗ *vme*, int ∗ *dmode*)

Get current Data mode.

Parameters:

∗***vme*** VME structure

∗***dmode*** returned address modifier

Returns:

MVME_SUCCESS

2.2.1.5 int EXPRT mvme_interrupt_attach (MVME_INTERFACE ∗ *mvme*, int *level*, int *vector*, void(∗)(int, void ∗, void ∗) *isr*, void ∗ *info*)

2.2.1.6 int EXPRT mvme_interrupt_detach (MVME_INTERFACE ∗ *mvme*, int *level*, int *vector*, void ∗ *info*)

2.2.1.7 int EXPRT mvme_interrupt_disable (MVME_INTERFACE ∗ *mvme*, int *level*, int *vector*, void ∗ *info*)

2.2.1.8 int EXPRT mvme_interrupt_enable (MVME_INTERFACE ∗ *mvme*, int *level*, int *vector*, void ∗ *info*)

2.2.1.9 int EXPRT mvme_interrupt_generate (MVME_INTERFACE ∗ *mvme*, int *level*, int *vector*, void ∗ *info*)

2.2.1.10 int EXPRT mvme_open (MVME_INTERFACE ∗∗ *vme*, int *idx*)

VME open The code below summarize the use of most of the mvme calls included in this interface.

```
#include "vmicvme.h"  // or other VME interface driver.

int main () {
  int i, status, vmeio_status, data;
  MVME_INTERFACE *myvme;

  // Open VME channel
  status = mvme_open(&myvme, 0);

  // Reset VME
  // Under VMIC reboot CPU!!
  //  status = mvme_sysreset(myvme);

  // Setup AM
  status = mvme_set_am(myvme, MVME_AM_A24_ND);

  // Setup Data size
  status = mvme_set_dmode(myvme, MVME_DMODE_D32);

  // Read VMEIO status
  status = mvme_read(myvme, &vmeio_status, 0x78001C, 4);
  printf("VMEIO status : 0x%x\n", vmeio_status);

  // Write Single value
  mvme_write_value(myvme, 0x780010, 0x3);

  // Read Single Value
  printf("Value : 0x%x\n", mvme_read_value(myvme, 0x780018));

  // Write to the VMEIO in latch mode
  for (i=0;i<10000;i++) {
    data = 0xF;
    status = mvme_write(myvme, 0x780010, &data, 4);
    data = 0x0;
    status = mvme_write(myvme, 0x780010, &data, 4);
  }

  // Close VME channel
  status = mvme_close(myvme);
  return 1;
}
```

Parameters:

- ∗∗***vme*** user VME pointer to the interface
- ***idx*** interface number should be used to distingush multiple VME interface access within the same program.

Returns:

status MVME_SUCCESS, MVME_NO_INTERFACE, MVME_INVALID_PARAM, MVME_ACCESS_ERROR

2.2.1.11 int EXPRT mvme_read (MVME_INTERFACE ∗ *vme*, void ∗ *dst*, mvme_addr_t *vme_addr*, mvme_size_t *n_bytes*)

Read from VME bus. Implementation of the read can include automatic DMA transfer based on the size of the data. See example in mvme_open()

Parameters:

- ∗***vme*** VME structure
- ∗***dst*** destination pointer
- ***vme_addr*** source address (VME location).
- ***n_bytes*** requested transfer size.

Returns:

MVME_SUCCESS

2.2.1.12 unsigned int EXPRT mvme_read_value (MVME_INTERFACE ∗ *vme*, mvme_addr_t *vme_addr*)

Read single data from VME bus. Useful for register access. See example in mvme_open()

Parameters:

- ∗***vme*** VME structure

vme_addr source address (VME location).

Returns:

MVME_SUCCESS

2.2.1.13 int EXPRT mvme_set_am (MVME_INTERFACE ∗ *vme*, int *am*)

Set Address Modifier.

Parameters:

∗***vme*** VME structure

am address modifier

Returns:

MVME_SUCCESS

2.2.1.14 int EXPRT mvme_set_blt (MVME_INTERFACE ∗ *vme*, int *mode*)

Set Block Transfer mode.

Parameters:

∗***vme*** VME structure

mode BLT mode

Returns:

MVME_SUCCESS

2.2.1.15 int EXPRT mvme_set_dmode (MVME_INTERFACE ∗ *vme*, int *dmode*)

Set Data mode.

Parameters:

∗***vme*** VME structure

dmode Data mode

Returns:

MVME_SUCCESS

2.2.1.16 int EXPRT mvme_sysreset (MVME_INTERFACE ∗ *vme*)

VME bus reset. Effect of the VME bus reset is dependent of the type of VME interface used. See example in mvme_open()

Parameters:

∗***vme*** VME structure.

Returns:

MVME_SUCCESS, MVME_ACCESS_ERROR

2.2.1.17 int EXPRT mvme_write (MVME_INTERFACE ∗ *vme*, mvme_addr_t *vme_addr*, void ∗ *src*, mvme_size_t *n_bytes*)

Write data to VME bus. Implementation of the write can include automatic DMA transfer based on the size of the data. See example in mvme_open()

Parameters:

∗***vme*** VME structure

vme_addr source address (VME location).

∗***src*** source array

n_bytes size of the array in bytes

Returns:

MVME_SUCCESS

2.2.1.18 int EXPRT mvme_write_value (MVME_INTERFACE ∗ *vme*, mvme_addr_t *vme_addr*, unsigned int *value*)

Write single data to VME bus. Useful for register access. See example in mvme_open()

Parameters:

∗***vme*** VME structure

vme_addr source address (VME location).

value Value to be written to the VME bus

Returns:

MVME_SUCCESS

2.3 Midas CAMAC standard

Modules

- group Camac Functions (camxxx)

2.4 Camac Functions (camxxx)

Functions

- EXTERNAL INLINE void EXPRT cam16i (const int c, const int n, const int a, const int f, WORD ∗d)
- EXTERNAL INLINE void EXPRT cam24i (const int c, const int n, const int a, const int f, DWORD ∗d)
- EXTERNAL INLINE void EXPRT cam8i_q (const int c, const int n, const int a, const int f, BYTE ∗d, int ∗x, int ∗q)
- EXTERNAL INLINE void EXPRT cam16i_q (const int c, const int n, const int a, const int f, WORD ∗d, int ∗x, int ∗q)
- EXTERNAL INLINE void EXPRT cam24i_q (const int c, const int n, const int a, const int f, DWORD ∗d, int ∗x, int ∗q)
- EXTERNAL INLINE void EXPRT cam16i_r (const int c, const int n, const int a, const int f, WORD ∗∗d, const int r)
- EXTERNAL INLINE void EXPRT cam24i_r (const int c, const int n, const int a, const int f, DWORD ∗∗d, const int r)
- EXTERNAL INLINE void EXPRT cam8i_rq (const int c, const int n, const int a, const int f, BYTE ∗∗d, const int r)
- EXTERNAL INLINE void EXPRT cam16i_rq (const int c, const int n, const int a, const int f, WORD ∗∗d, const int r)
- EXTERNAL INLINE void EXPRT cam24i_rq (const int c, const int n, const int a, const int f, DWORD ∗∗d, const int r)
- EXTERNAL INLINE void EXPRT cam8i_sa (const int c, const int n, const int a, const int f, BYTE ∗∗d, const int r)
- EXTERNAL INLINE void EXPRT cam16i_sa (const int c, const int n, const int a, const int f, WORD ∗∗d, const int r)
- EXTERNAL INLINE void EXPRT cam24i_sa (const int c, const int n, const int a, const int f, DWORD ∗∗d, const int r)
- EXTERNAL INLINE void EXPRT cam8i_sn (const int c, const int n, const int a, const int f, BYTE ∗∗d, const int r)
- EXTERNAL INLINE void EXPRT cam16i_sn (const int c, const int n, const int a, const int f, WORD ∗∗d, const int r)
- EXTERNAL INLINE void EXPRT cam24i_sn (const int c, const int n, const int a, const int f, DWORD ∗∗d, const int r)

- EXTERNAL INLINE void EXPRT cami (const int c, const int n, const int a, const int f, WORD ∗d)
- EXTERNAL INLINE void EXPRT cam8o (const int c, const int n, const int a, const int f, BYTE d)
- EXTERNAL INLINE void EXPRT cam16o (const int c, const int n, const int a, const int f, WORD d)
- EXTERNAL INLINE void EXPRT cam24o (const int c, const int n, const int a, const int f, DWORD d)
- EXTERNAL INLINE void EXPRT cam8o_q (const int c, const int n, const int a, const int f, BYTE d, int ∗x, int ∗q)
- EXTERNAL INLINE void EXPRT cam16o_q (const int c, const int n, const int a, const int f, WORD d, int ∗x, int ∗q)
- EXTERNAL INLINE void EXPRT cam24o_q (const int c, const int n, const int a, const int f, DWORD d, int ∗x, int ∗q)
- EXTERNAL INLINE void EXPRT cam8o_r (const int c, const int n, const int a, const int f, BYTE ∗d, const int r)
- EXTERNAL INLINE void EXPRT cam16o_r (const int c, const int n, const int a, const int f, WORD ∗d, const int r)
- EXTERNAL INLINE void EXPRT cam24o_r (const int c, const int n, const int a, const int f, DWORD ∗d, const int r)
- EXTERNAL INLINE void EXPRT camo (const int c, const int n, const int a, const int f, WORD d)
- EXTERNAL INLINE int EXPRT camc_chk (const int c)
- EXTERNAL INLINE void EXPRT camc (const int c, const int n, const int a, const int f)
- EXTERNAL INLINE void EXPRT camc_q (const int c, const int n, const int a, const int f, int ∗q)
- EXTERNAL INLINE void EXPRT camc_sa (const int c, const int n, const int a, const int f, const int r)
- EXTERNAL INLINE void EXPRT camc_sn (const int c, const int n, const int a, const int f, const int r)
- EXTERNAL INLINE int EXPRT cam_init (void)
- EXTERNAL INLINE int EXPRT cam_init_rpc (char ∗host_name, char ∗exp_name, char ∗fe_name, char ∗client_name, char ∗rpc_server)
- EXTERNAL INLINE void EXPRT cam_exit (void)
- EXTERNAL INLINE void EXPRT cam_inhibit_set (const int c)
- EXTERNAL INLINE void EXPRT cam_inhibit_clear (const int c)
- EXTERNAL INLINE int EXPRT cam_inhibit_test (const int c)
- EXTERNAL INLINE void EXPRT cam_crate_clear (const int c)
- EXTERNAL INLINE void EXPRT cam_crate_zinit (const int c)
- EXTERNAL INLINE void EXPRT cam_lam_enable (const int c, const int n)
- EXTERNAL INLINE void EXPRT cam_lam_disable (const int c, const int n)
- EXTERNAL void cam_lam_read (const int c, DWORD ∗lam)
- EXTERNAL INLINE void EXPRT cam_lam_clear (const int c, const int n)

- EXTERNAL INLINE int EXPRT cam_lam_wait (int ∗c, DWORD ∗n, const int millisec)
- EXTERNAL INLINE void EXPRT cam_interrupt_enable (const int c)
- EXTERNAL INLINE void EXPRT cam_interrupt_disable (const int c)
- EXTERNAL INLINE int EXPRT cam_interrupt_test (const int c)
- EXTERNAL INLINE void EXPRT cam_interrupt_attach (const int c, const int n, void(∗isr)(void))
- EXTERNAL INLINE void EXPRT cam_interrupt_detach (const int c, const int n)

2.4.1 Function Documentation

2.4.1.1 EXTERNAL INLINE void EXPRT cam16i (const int *c*, const int *n*, const int *a*, const int *f*, WORD ∗ *d*)

16 bits input.

Parameters:

c crate number (0..)

n station number (0..30)

a sub-address (0..15)

f function (0..7)

d data read out data

Returns:

void

2.4.1.2 EXTERNAL INLINE void EXPRT cam16i_q (const int *c*, const int *n*, const int *a*, const int *f*, WORD ∗ *d*, int ∗ *x*, int ∗ *q*)

16 bits input with Q response.

Parameters:

c crate number (0..)

n station number (0..30)

a sub-address (0..15)

f function (0..7)

d data read out data

x X response (0:failed,1:success)

q Q resonpse (0:no Q, 1: Q)

Returns:

void

Referenced by csmad(), and cssa().

2.4.1.3 EXTERNAL INLINE void EXPRT cam16i_r (const int *c*, const int *n*, const int *a*, const int *f*, WORD ∗∗ *d*, const int *r*)

Repeat 16 bits input.

Parameters:

c crate number (0..)

n station number (0..30)

a sub-address (0..15)

f function (0..7)

d data read out data

r repeat time

Returns:

void

2.4.1.4 EXTERNAL INLINE void EXPRT cam16i_rq (const int *c*, const int *n*, const int *a*, const int *f*, WORD ∗∗ *d*, const int *r*)

Repeat 16 bits input with Q stop.

Parameters:

c crate number (0..)

n station number (0..30)

a sub-address (0..15)

f function (0..7)

d pointer to data read out

r repeat time

Returns:

void

2.4.1.5 EXTERNAL INLINE void EXPRT cam16i_sa (const int *c*, const int *n*, const int *a*, const int *f*, WORD ∗∗ *d*, const int *r*)

Read the given CAMAC address and increment the sub-address by one. Repeat r times.

```
WORD pbkdat[4];
cam16i_sa(crate, 5, 0, 2, &pbkdat, 4);
```

equivalent to :

```
cam16i(crate, 5, 0, 2, &pbkdat[0]);
cam16i(crate, 5, 1, 2, &pbkdat[1]);
cam16i(crate, 5, 2, 2, &pbkdat[2]);
cam16i(crate, 5, 3, 2, &pbkdat[3]);
```

Parameters:

c crate number (0..)

n station number (0..30)

a sub-address (0..15)

f function (0..7)

d pointer to data read out

r number of consecutive sub-address to read

Returns:

void

2.4.1.6 EXTERNAL INLINE void EXPRT cam16i_sn (const int *c*, const int *n*, const int *a*, const int *f*, WORD ∗∗ *d*, const int *r*)

Read the given CAMAC address and increment the station number by one. Repeat r times.

```
WORD pbkdat[4];
cam16i_sa(crate, 5, 0, 2, &pbkdat, 4);
```

equivalent to :

```
cam16i(crate, 5, 0, 2, &pbkdat[0]);
cam16i(crate, 6, 0, 2, &pbkdat[1]);
cam16i(crate, 7, 0, 2, &pbkdat[2]);
cam16i(crate, 8, 0, 2, &pbkdat[3]);
```

Parameters:

c crate number (0..)

- ***n*** station number (0..30)
- ***a*** sub-address (0..15)
- ***f*** function (0..7)
- ***d*** pointer to data read out
- ***r*** number of consecutive station to read

Returns:
void

2.4.1.7 EXTERNAL INLINE void EXPRT cam16o (const int *c*, const int *n*, const int *a*, const int *f*, WORD *d*)

Write data to given CAMAC address.

Parameters:

- ***c*** crate number (0..)
- ***n*** station number (0..30)
- ***a*** sub-address (0..15)
- ***f*** function (16..31)
- ***d*** data to be written to CAMAC

Returns:
void

2.4.1.8 EXTERNAL INLINE void EXPRT cam16o_q (const int *c*, const int *n*, const int *a*, const int *f*, WORD *d*, int ∗ *x*, int ∗ *q*)

Write data to given CAMAC address with Q response.

Parameters:

- ***c*** crate number (0..)
- ***n*** station number (0..30)
- ***a*** sub-address (0..15)
- ***f*** function (16..31)
- ***d*** data to be written to CAMAC
- ***x*** X response (0:failed,1:success)
- ***q*** Q resonpse (0:no Q, 1: Q)

Returns:
void

Referenced by cssa().

2.4.1.9 EXTERNAL INLINE void EXPRT cam16o_r (const int *c*, const int *n*, const int *a*, const int *f*, WORD ∗ *d*, const int *r*)

Repeat write data to given CAMAC address r times.

Parameters:

c crate number (0..)

n station number (0..30)

a sub-address (0..15)

f function (16..31)

d data to be written to CAMAC

r number of repeatition

Returns:
void

2.4.1.10 EXTERNAL INLINE void EXPRT cam24i (const int *c*, const int *n*, const int *a*, const int *f*, DWORD ∗ *d*)

24 bits input.

Parameters:

c crate number (0..)

n station number (0..30)

a sub-address (0..15)

f function (0..7)

d data read out data

Returns:
void

Referenced by read_scaler_event().

2.4.1.11 EXTERNAL INLINE void EXPRT cam24i_q (const int *c*, const int *n*, const int *a*, const int *f*, DWORD ∗ *d*, int ∗ *x*, int ∗ *q*)

24 bits input with Q response.

Parameters:

- ***c*** crate number (0..)
- ***n*** station number (0..30)
- ***a*** sub-address (0..15)
- ***f*** function (0..7)
- ***d*** data read out data
- ***x*** X response (0:failed,1:success)
- ***q*** Q resonpse (0:no Q, 1: Q)

Returns:

void

Referenced by cfmad(), and cfsa().

2.4.1.12 EXTERNAL INLINE void EXPRT cam24i_r (const int *c*, const int *n*, const int *a*, const int *f*, DWORD ∗∗ *d*, const int *r*)

Repeat 24 bits input.

Parameters:

- ***c*** crate number (0..)
- ***n*** station number (0..30)
- ***a*** sub-address (0..15)
- ***f*** function (0..7)
- ***d*** data read out
- ***r*** repeat time

Returns:

void

2.4.1.13 EXTERNAL INLINE void EXPRT cam24i_rq (const int *c*, const int *n*, const int *a*, const int *f*, DWORD ∗∗ *d*, const int *r*)

Repeat 24 bits input with Q stop.

Parameters:

- ***c*** crate number (0..)
- ***n*** station number (0..30)
- ***a*** sub-address (0..15)
- ***f*** function (0..7)
- ***d*** pointer to data read out
- ***r*** repeat time

Returns:

void

2.4.1.14 EXTERNAL INLINE void EXPRT cam24i_sa (const int *c*, const int *n*, const int *a*, const int *f*, DWORD $**$ *d*, const int *r*)

Read the given CAMAC address and increment the sub-address by one. Repeat r times.

```
DWORD pbkdat[8];
cam24i_sa(crate, 5, 0, 2, &pbkdat, 8);
```

equivalent to

```
cam24i(crate, 5, 0, 2, &pbkdat[0]);
cam24i(crate, 6, 0, 2, &pbkdat[1]);
cam24i(crate, 7, 0, 2, &pbkdat[2]);
cam24i(crate, 8, 0, 2, &pbkdat[3]);
```

Parameters:

- ***c*** crate number (0..)
- ***n*** station number (0..30)
- ***a*** sub-address (0..15)
- ***f*** function (0..7)
- ***d*** pointer to data read out
- ***r*** number of consecutive sub-address to read

Returns:

void

2.4.1.15 EXTERNAL INLINE void EXPRT cam24i_sn (const int *c*, const int *n*, const int *a*, const int *f*, DWORD $**$ *d*, const int *r*)

Read the given CAMAC address and increment the station number by one. Repeat r times.

```
DWORD pbkdat[4];
cam24i_sa(crate, 5, 0, 2, &pbkdat, 4);
```

equivalent to :

```
cam24i(crate, 5, 0, 2, &pbkdat[0]);
cam24i(crate, 6, 0, 2, &pbkdat[1]);
cam24i(crate, 7, 0, 2, &pbkdat[2]);
cam24i(crate, 8, 0, 2, &pbkdat[3]);
```

Parameters:

c crate number (0..)

n station number (0..30)

a sub-address (0..15)

f function (0..7)

d pointer to data read out

r number of consecutive station to read

Returns:

void

2.4.1.16 EXTERNAL INLINE void EXPRT cam24o (const int *c*, const int *n*, const int *a*, const int *f*, DWORD *d*)

Write data to given CAMAC address.

Parameters:

c crate number (0..)

n station number (0..30)

a sub-address (0..15)

f function (16..31)

d data to be written to CAMAC

Returns:

void

2.4.1.17 EXTERNAL INLINE void EXPRT cam24o_q (const int *c*, const int *n*, const int *a*, const int *f*, DWORD *d*, int ∗ *x*, int ∗ *q*)

Write data to given CAMAC address with Q response.

Parameters:

- ***c*** crate number (0..)
- ***n*** station number (0..30)
- ***a*** sub-address (0..15)
- ***f*** function (16..31)
- ***d*** data to be written to CAMAC
- ***x*** X response (0:failed,1:success)
- ***q*** Q response (0:no Q, 1: Q)

Returns:

void

Referenced by cfsa().

2.4.1.18 EXTERNAL INLINE void EXPRT cam24o_r (const int *c*, const int *n*, const int *a*, const int *f*, DWORD ∗ *d*, const int *r*)

Repeat write data to given CAMAC address r times.

Parameters:

- ***c*** crate number (0..)
- ***n*** station number (0..30)
- ***a*** sub-address (0..15)
- ***f*** function (16..31)
- ***d*** data to be written to CAMAC
- ***r*** number of repeatition

Returns:

void

2.4.1.19 EXTERNAL INLINE void EXPRT cam8i_q (const int *c*, const int *n*, const int *a*, const int *f*, BYTE ∗ *d*, int ∗ *x*, int ∗ *q*)

8 bits input with Q response.

Parameters:

- ***c*** crate number (0..)
- ***n*** station number (0..30)
- ***a*** sub-address (0..15)
- ***f*** function (0..7)
- ***d*** data read out data
- ***x*** X response (0:failed,1:success)
- ***q*** Q resonpse (0:no Q, 1: Q)

Returns:

void

2.4.1.20 EXTERNAL INLINE void EXPRT cam8i_rq (const int *c*, const int *n*, const int *a*, const int *f*, BYTE ∗∗ *d*, const int *r*)

Repeat 8 bits input with Q stop.

Parameters:

- ***c*** crate number (0..)
- ***n*** station number (0..30)
- ***a*** sub-address (0..15)
- ***f*** function (0..7)
- ***d*** pointer to data read out
- ***r*** repeat time

Returns:

void

2.4.1.21 EXTERNAL INLINE void EXPRT cam8i_sa (const int *c*, const int *n*, const int *a*, const int *f*, BYTE ∗∗ *d*, const int *r*)

Read the given CAMAC address and increment the sub-address by one. Repeat r times.

```
BYTE pbkdat[4];
cam8i_sa(crate, 5, 0, 2, &pbkdat, 4);
```

equivalent to :

```
cam8i(crate, 5, 0, 2, &pbkdat[0]);
cam8i(crate, 5, 1, 2, &pbkdat[1]);
cam8i(crate, 5, 2, 2, &pbkdat[2]);
cam8i(crate, 5, 3, 2, &pbkdat[3]);
```

Parameters:

- ***c*** crate number (0..)
- ***n*** station number (0..30)
- ***a*** sub-address (0..15)
- ***f*** function (0..7)
- ***d*** pointer to data read out
- ***r*** number of consecutive sub-address to read

Returns:

void

2.4.1.22 EXTERNAL INLINE void EXPRT cam8i_sn (const int *c*, const int *n*, const int *a*, const int *f*, BYTE ∗∗ *d*, const int *r*)

Read the given CAMAC address and increment the station number by one. Repeat r times.

```
BYTE pbkdat[4];
cam8i_sa(crate, 5, 0, 2, &pbkdat, 4);
```

equivalent to :

```
cam8i(crate, 5, 0, 2, &pbkdat[0]);
cam8i(crate, 6, 0, 2, &pbkdat[1]);
cam8i(crate, 7, 0, 2, &pbkdat[2]);
cam8i(crate, 8, 0, 2, &pbkdat[3]);
```

Parameters:

- ***c*** crate number (0..)
- ***n*** station number (0..30)
- ***a*** sub-address (0..15)
- ***f*** function (0..7)
- ***d*** pointer to data read out
- ***r*** number of consecutive station to read

Returns:

void

2.4.1.23 EXTERNAL INLINE void EXPRT cam8o (const int *c*, const int *n*, const int *a*, const int *f*, BYTE *d*)

Write data to given CAMAC address.

Parameters:

- ***c*** crate number (0..)
- ***n*** station number (0..30)
- ***a*** sub-address (0..15)
- ***f*** function (16..31)
- ***d*** data to be written to CAMAC

Returns:

void

2.4.1.24 EXTERNAL INLINE void EXPRT cam8o_q (const int *c*, const int *n*, const int *a*, const int *f*, BYTE *d*, int $*$ *x*, int $*$ *q*)

Write data to given CAMAC address with Q response.

Parameters:

- ***c*** crate number (0..)
- ***n*** station number (0..30)
- ***a*** sub-address (0..15)
- ***f*** function (16..31)
- ***d*** data to be written to CAMAC
- ***x*** X response (0:failed,1:success)
- ***q*** Q resonpse (0:no Q, 1: Q)

Returns:

void

2.4.1.25 EXTERNAL INLINE void EXPRT cam8o_r (const int *c*, const int *n*, const int *a*, const int *f*, BYTE $*$ *d*, const int *r*)

Repeat write data to given CAMAC address r times.

Parameters:

- ***c*** crate number (0..)

n station number (0..30)

a sub-address (0..15)

f function (16..31)

d data to be written to CAMAC

r number of repeatition

Returns:

void

2.4.1.26 EXTERNAL INLINE void EXPRT cam_crate_clear (const int *c*)

Issue CLEAR to crate.

Parameters:

c crate number (0..)

Returns:

void

Referenced by cccc(), and frontend_init().

2.4.1.27 EXTERNAL INLINE void EXPRT cam_crate_zinit (const int *c*)

Issue Z to crate.

Parameters:

c crate number (0..)

Returns:

void

Referenced by cccz(), and frontend_init().

2.4.1.28 EXTERNAL INLINE void EXPRT cam_exit (void)

Close CAMAC accesss.

2.4.1.29 EXTERNAL INLINE void EXPRT cam_inhibit_clear (const int *c*)

Clear Crate inhibit.

Parameters:

c crate number (0..)

Returns:

void

Referenced by ccci().

2.4.1.30 EXTERNAL INLINE void EXPRT cam_inhibit_set (const int *c*)

Set Crate inhibit.

Parameters:

c crate number (0..)

Returns:

void

Referenced by ccci().

2.4.1.31 EXTERNAL INLINE int EXPRT cam_inhibit_test (const int *c*)

Test Crate Inhibit.

Parameters:

c crate number (0..)

Returns:

1 for set, 0 for cleared

Referenced by ctci().

2.4.1.32 EXTERNAL INLINE int EXPRT cam_init (void)

Initialize CAMAC access.

Returns:

1: success

Referenced by ccinit(), fccinit(), and frontend_init().

2.4.1.33 EXTERNAL INLINE int EXPRT cam_init_rpc (char ∗ *host_name*, char ∗ *exp_name*, char ∗ *fe_name*, char ∗ *client_name*, char ∗ *rpc_server*)

Initialize CAMAC access for rpc calls

For Internal use only.

Parameters:

- ***host_name*** Midas host to contact
- ***exp_name*** Midas experiment to contact
- ***fe_name*** frontend application name to contact
- ***client_name*** RPC host name
- ***rpc_server*** RPC server name

Returns:

1: success

2.4.1.34 EXTERNAL INLINE void EXPRT cam_interrupt_attach (const int *c*, const int *n*, void(∗)(void) *isr*)

Attach service routine to LAM of specific crate and station.

Parameters:

- ***c*** crate number (0..)
- ***n*** station number
- ***(∗isr)*** Function pointer to attach to the LAM

Returns:

void

Referenced by cclnk().

2.4.1.35 EXTERNAL INLINE void EXPRT cam_interrupt_detach (const int *c*, const int *n*)

Detach service routine from LAM.

Parameters:

- ***c*** crate number (0..)
- ***n*** station number

Returns:

void

Referenced by cculk().

2.4.1.36 EXTERNAL INLINE void EXPRT cam_interrupt_disable (const int *c*)

Disables interrupts in specific crate

Parameters:

c crate number (0..)

Returns:

void

Referenced by cccd().

2.4.1.37 EXTERNAL INLINE void EXPRT cam_interrupt_enable (const int *c*)

Enable interrupts in specific crate

Parameters:

c crate number (0..)

Returns:

void

Referenced by cccd(), and ccrgl().

2.4.1.38 EXTERNAL INLINE int EXPRT cam_interrupt_test (const int *c*)

Test Crate Interrupt.

Parameters:

c crate number (0..)

Returns:

1 for set, 0 for cleared

Referenced by ctcd().

2.4.1.39 EXTERNAL INLINE void EXPRT cam_lam_clear (const int *c*, const int *n*)

Clear the LAM register of the crate controller. It doesn't clear the LAM of the particular station.

Parameters:

c crate number (0..)

n LAM station

Returns:

void

Referenced by cclnk(), ccrgl(), and read_trigger_event().

2.4.1.40 EXTERNAL INLINE void EXPRT cam_lam_disable (const int *c*, const int *n*)

Disable LAM generation for given station to the Crate controller. It doesn't disable the LAM of the actual station itself.

Parameters:

c crate number (0..)

n LAM station

Returns:

void

2.4.1.41 EXTERNAL INLINE void EXPRT cam_lam_enable (const int *c*, const int *n*)

Enable LAM generation for given station to the Crate controller. It doesn't enable the LAM of the actual station itself.

Parameters:

c crate number (0..)

n LAM station

Returns:

void

Referenced by cclnk(), ccrgl(), and frontend_init().

2.4.1.42 EXTERNAL void cam_lam_read (const int *c*, DWORD ∗ *lam*)

Reads in lam the lam pattern of the entire crate.

Parameters:

c crate number (0..)

lam LAM pattern of the crate

Returns:
void

Referenced by ctgl(), and poll_event().

2.4.1.43 EXTERNAL INLINE int EXPRT cam_lam_wait (int ∗ *c*, DWORD ∗ *n*, const int *millisec*)

Wait for a LAM to occur with a certain timeout. Return crate and station if LAM occurs.

Parameters:

c crate number (0..)

n LAM station

millisec If there is no LAM after this timeout, the routine returns

Returns:
1 if LAM occured, 0 else

2.4.1.44 EXTERNAL INLINE void EXPRT camc (const int *c*, const int *n*, const int *a*, const int *f*)

CAMAC command (no data).

Parameters:

c crate number (0..)

n station number (0..30)

a sub-address (0..15)

f function (8..15, 24..31)

Returns:
void

Referenced by cclc(), cclm(), frontend_init(), and read_trigger_event().

2.4.1.45 EXTERNAL INLINE int EXPRT camc_chk (const int *c*)

Crate presence check.

Parameters:

c crate number (0..)

Returns:
0:Success, -1:No CAMAC response

2.4.1.46 EXTERNAL INLINE void EXPRT camc_q (const int *c*, const int *n*, const int *a*, const int *f*, int ∗ *q*)

CAMAC command with Q response (no data).

Parameters:
- ***c*** crate number (0..)
- ***n*** station number (0..30)
- ***a*** sub-address (0..15)
- ***f*** function (8..15, 24..31)
- ***q*** Q response (0:no Q, 1:Q)

Returns:
void

Referenced by cfsa(), cssa(), ctlm(), and read_trigger_event().

2.4.1.47 EXTERNAL INLINE void EXPRT camc_sa (const int *c*, const int *n*, const int *a*, const int *f*, const int *r*)

Scan CAMAC command on sub-address.

Parameters:
- ***c*** crate number (0..)
- ***n*** station number (0..30)
- ***a*** sub-address (0..15)
- ***f*** function (8..15, 24..31)
- ***r*** number of consecutive sub-address to read

Returns:
void

2.4.1.48 EXTERNAL INLINE void EXPRT camc_sn (const int *c*, const int *n*, const int *a*, const int *f*, const int *r*)

Scan CAMAC command on station.

Parameters:

- ***c*** crate number (0..)
- ***n*** station number (0..30)
- ***a*** sub-address (0..15)
- ***f*** function (8..15, 24..31)
- ***r*** number of consecutive station to read

Returns:

void

2.4.1.49 EXTERNAL INLINE void EXPRT cami (const int *c*, const int *n*, const int *a*, const int *f*, WORD $*$ *d*)

Same as cam16i()

2.4.1.50 EXTERNAL INLINE void EXPRT camo (const int *c*, const int *n*, const int *a*, const int *f*, WORD *d*)

Same as cam16o()

Referenced by frontend_init(), and read_trigger_event().

2.4.1.51 EXTERNAL INLINE void EXPRT camop ()

Definition at line 744 of file mcstd.h.

2.5 The midas.h & midas.c

Modules

- group Midas Define
- group Midas Macros
- group Midas Error definition
- group Midas Structure Declaration
- group Midas Message Functions (msg_xxx)
- group Midas Common Functions (cm_xxx)
- group Midas Buffer Manager Functions (bm_xxx)

- group Midas RPC Functions (rpc_xxx)
- group Midas Bank Functions (bk_xxx)
- group Midas Dual Buffer Memory Functions (dm_xxx)
- group Midas Ring Buffer Functions (rb_xxx)

Defines

- #define MAX_EVENT_SIZE 0x400000
- #define TAPE_BUFFER_SIZE 0x8000
- #define NET_TCP_SIZE 0xFFFF
- #define OPT_TCP_SIZE 8192
- #define NET_UDP_SIZE 8192
- #define EVENT_BUFFER_NAME "SYSTEM"
- #define DEFAULT_ODB_SIZE 0x100000
- #define NAME_LENGTH 32
- #define HOST_NAME_LENGTH 256
- #define MAX_CLIENTS 64
- #define MAX_EVENT_REQUESTS 10
- #define MAX_OPEN_RECORDS 256
- #define MAX_ODB_PATH 256
- #define MAX_EXPERIMENT 32
- #define BANKLIST_MAX 64
- #define STRING_BANKLIST_MAX BANKLIST_MAX ∗ 4
- #define DEFAULT_RPC_TIMEOUT 10000
- #define DEFAULT_WATCHDOG_TIMEOUT 10000
- #define CH_BS 8
- #define LAM_SOURCE(c, s) (c<<24 | ((s) & 0xFFFFFF))
- #define LAM_STATION(s) (1<<(s-1))
- #define LAM_SOURCE_CRATE(c) (c>>24)
- #define LAM_SOURCE_STATION(s) ((s) & 0xFFFFFF)
- #define CNAF 0x1
- #define ANA_CONTINUE 1

Variables

- HNDLE _hKeyClient = 0

2.5.1 Define Documentation

2.5.1.1 #define ANA_CONTINUE 1

dox***

Definition at line 739 of file midas.h.

2.5.1.2 #define ANA_SKIP 0

Definition at line 740 of file midas.h.

2.5.1.3 #define BANKLIST_MAX 64

max # of banks in event

Definition at line 230 of file midas.h.

Referenced by bk_list().

2.5.1.4 #define CH_BS 8

special characters

Definition at line 371 of file midas.h.

2.5.1.5 #define CH_CR 13

Definition at line 373 of file midas.h.

2.5.1.6 #define CH_DELETE (CH_EXT+2)

Definition at line 379 of file midas.h.

2.5.1.7 #define CH_DOWN (CH_EXT+7)

Definition at line 384 of file midas.h.

2.5.1.8 #define CH_END (CH_EXT+3)

Definition at line 380 of file midas.h.

2.5.1.9 #define CH_EXT 0x100

Definition at line 375 of file midas.h.

2.5.1.10 #define CH_HOME (CH_EXT+0)

Definition at line 377 of file midas.h.

2.5.1.11 #define CH_INSERT (CH_EXT+1)

Definition at line 378 of file midas.h.

2.5.1.12 #define CH_LEFT (CH_EXT+9)

Definition at line 386 of file midas.h.

2.5.1.13 #define CH_PDOWN (CH_EXT+5)

Definition at line 382 of file midas.h.

2.5.1.14 #define CH_PUP (CH_EXT+4)

Definition at line 381 of file midas.h.

2.5.1.15 #define CH_RIGHT (CH_EXT+8)

Definition at line 385 of file midas.h.

2.5.1.16 #define CH_TAB 9

Definition at line 372 of file midas.h.

2.5.1.17 #define CH_UP (CH_EXT+6)

Definition at line 383 of file midas.h.

2.5.1.18 #define CNAF 0x1

CNAF commands

Definition at line 416 of file midas.h.

2.5.1.19 #define CNAF_CRATE_CLEAR 0x102

Definition at line 421 of file midas.h.

2.5.1.20 #define CNAF_CRATE_ZINIT 0x103

Definition at line 422 of file midas.h.

2.5.1.21 #define CNAF_INHIBIT_CLEAR 0x101

Definition at line 420 of file midas.h.

2.5.1.22 #define CNAF_INHIBIT_SET 0x100

Definition at line 419 of file midas.h.

2.5.1.23 #define CNAF_nQ 0x2

Definition at line 417 of file midas.h.

2.5.1.24 #define CNAF_TEST 0x110

Definition at line 423 of file midas.h.

2.5.1.25 #define DATABASE_VERSION 2

Definition at line 41 of file midas.h.

Referenced by db_open_database().

2.5.1.26 #define DEFAULT_ODB_SIZE 0x100000

online database 1M

Definition at line 221 of file midas.h.

Referenced by cm_connect_experiment(), and main().

2.5.1.27 #define DEFAULT_RPC_TIMEOUT 10000

Timeouts [ms]

Definition at line 238 of file midas.h.

2.5.1.28 #define DEFAULT_WATCHDOG_TIMEOUT 10000

Watchdog

Definition at line 241 of file midas.h.

Referenced by cm_connect_experiment().

2.5.1.29 #define EVENT_BUFFER_NAME "SYSTEM"

buffer name for commands

Definition at line 220 of file midas.h.

2.5.1.30 #define HOST_NAME_LENGTH 256

length of TCP/IP names

Definition at line 224 of file midas.h.

Referenced by cm_set_client_info().

2.5.1.31 #define LAM_SOURCE(c, s) (c<<24 | ((s) & 0xFFFFFF))

Code the LAM crate and LAM station into a bitwise register.

Parameters:

- ***c*** Crate number
- ***s*** Slot number

Definition at line 394 of file midas.h.

2.5.1.32 #define LAM_SOURCE_CRATE(c) (c>>24)

Convert the coded LAM crate to Crate number.

Parameters:

- ***c*** coded crate

Definition at line 406 of file midas.h.

Referenced by poll_event().

2.5.1.33 #define LAM_SOURCE_STATION(s) ((s) & 0xFFFFFF)

Convert the coded LAM station to Station number.

Parameters:

- ***s*** Slot number

Definition at line 412 of file midas.h.

Referenced by poll_event().

2.5.1.34 #define LAM_STATION(s) (1<<(s-1))

Code the Station number bitwise for the LAM source.

Parameters:

- ***s*** Slot number

Definition at line 400 of file midas.h.

2.5.1.35 #define MAX_CLIENTS 64

client processes per buf/db

Definition at line 225 of file midas.h.

Referenced by bm_close_buffer(), cm_cleanup(), and db_close_database().

2.5.1.36 #define MAX_EVENT_REQUESTS 10

event requests per client

Definition at line 226 of file midas.h.

Referenced by bm_remove_event_request().

2.5.1.37 #define MAX_EVENT_SIZE 0x400000

maximum event size 4MB

Definition at line 208 of file midas.h.

Referenced by bm_send_event(), main(), register_equipment(), rpc_send_event(), and source_booking().

2.5.1.38 #define MAX_EXPERIMENT 32

number of different exp.

Definition at line 229 of file midas.h.

Referenced by cm_connect_experiment1(), and cm_list_experiments().

2.5.1.39 #define MAX_ODB_PATH 256

length of path in ODB

Definition at line 228 of file midas.h.

2.5.1.40 #define MAX_OPEN_RECORDS 256

number of open DB records

Definition at line 227 of file midas.h.

2.5.1.41 #define MIDAS_TCP_PORT 1175

Definition at line 234 of file midas.h.

Referenced by cm_list_experiments().

2.5.1.42 #define MIDAS_VERSION "2.0.0"

Definition at line 44 of file midas.h.

2.5.1.43 #define NAME_LENGTH 32

length of names, mult.of 8!

Definition at line 223 of file midas.h.

Referenced by cm_connect_experiment1(), cm_set_client_info(), and hs_dump().

2.5.1.44 #define NET_TCP_SIZE 0xFFFF

maximum TCP transfer size

Definition at line 216 of file midas.h.

Referenced by rpc_send_event(), and scheduler().

2.5.1.45 #define NET_UDP_SIZE 8192

maximum UDP transfer

Definition at line 218 of file midas.h.

2.5.1.46 #define OPT_TCP_SIZE 8192

optimal TCP buffer size

Definition at line 217 of file midas.h.

2.5.1.47 #define STRING_BANKLIST_MAX BANKLIST_MAX $*$ 4

for bk_list()

Definition at line 231 of file midas.h.

2.5.1.48 #define TAPE_BUFFER_SIZE 0x8000

buffer size for taping data

Definition at line 214 of file midas.h.

2.5.1.49 #define WATCHDOG_INTERVAL 1000

Definition at line 239 of file midas.h.

Referenced by cm_set_client_info(), and cm_set_watchdog_params().

2.5.2 Variable Documentation

2.5.2.1 INT _call_watchdog = TRUE `[static]`

Definition at line 1024 of file midas.c.

Referenced by cm_set_watchdog_params().

2.5.2.2 char _client_name[NAME_LENGTH] `[static]`

Definition at line 1022 of file midas.c.

2.5.2.3 HNDLE _hDB = 0 `[static]`

Definition at line 1021 of file midas.c.

Referenced by cm_set_experiment_database().

2.5.2.4 HNDLE _hKeyClient = 0 `[static]`

dox***

Definition at line 1020 of file midas.c.

Referenced by cm_set_experiment_database().

2.5.2.5 INT _mutex_alarm

Definition at line 1026 of file midas.c.

Referenced by db_close_database().

2.5.2.6 INT _mutex_elog

Definition at line 1026 of file midas.c.

Referenced by db_close_database().

2.5.2.7 INT _mutex_history

Definition at line 1026 of file midas.c.

2.5.2.8 char _path_name[MAX_STRING_LENGTH] `[static]`

Definition at line 1023 of file midas.c.

Referenced by cm_get_path(), and cm_set_path().

2.5.2.9 INT _watchdog_timeout = DEFAULT_WATCHDOG_TIMEOUT `[static]`

Definition at line 1025 of file midas.c.

Referenced by cm_set_watchdog_params().

2.6 Midas Define

Defines

- #define STATE_STOPPED 1
- #define STATE_PAUSED 2
- #define STATE_RUNNING 3
- #define FORMAT_MIDAS 1
- #define FORMAT_YBOS 2
- #define FORMAT_ASCII 3
- #define FORMAT_FIXED 4
- #define FORMAT_DUMP 5
- #define FORMAT_HBOOK 6
- #define FORMAT_ROOT 7
- #define GET_ALL (1<<0)
- #define GET_SOME (1<<1)
- #define GET_FARM (1<<2)
- #define TID_BYTE 1
- #define TID_SBYTE 2
- #define TID_CHAR 3
- #define TID_WORD 4
- #define TID_SHORT 5
- #define TID_DWORD 6
- #define TID_INT 7
- #define TID_BOOL 8
- #define TID_FLOAT 9
- #define TID_DOUBLE 10
- #define TID_BITFIELD 11
- #define TID_STRING 12
- #define TID_ARRAY 13
- #define TID_STRUCT 14
- #define TID_KEY 15
- #define TID_LINK 16
- #define TID_LAST 17

- #define SYNC 0
- #define MODE_READ (1<<0)
- #define RPC_OTIMEOUT 1
- #define WF_WATCH_ME (1<<0)
- #define TR_START (1<<0)
- #define TR_STOP (1<<1)
- #define TR_PAUSE (1<<2)
- #define TR_RESUME (1<<3)
- #define EQ_PERIODIC (1<<0)
- #define EQ_POLLED (1<<1)
- #define EQ_INTERRUPT (1<<2)
- #define EQ_MULTITHREAD (1<<3)
- #define EQ_SLOW (1<<4)
- #define EQ_MANUAL_TRIG (1<<5)
- #define EQ_FRAGMENTED (1<<6)
- #define EQ_EB (1<<7)
- #define RO_RUNNING (1<<0)
- #define RO_STOPPED (1<<1)
- #define RO_PAUSED (1<<2)
- #define RO_BOR (1<<3)
- #define RO_EOR (1<<4)
- #define RO_PAUSE (1<<5)
- #define RO_RESUME (1<<6)
- #define RO_TRANSITIONS (RO_BOR|RO_EOR|RO_PAUSE|RO_RESUME)
- #define RO_ALWAYS (0xFF)
- #define RO_ODB (1<<8)
- #define MT_ERROR (1<<0)
- #define MT_INFO (1<<1)
- #define MT_DEBUG (1<<2)
- #define MT_USER (1<<3)
- #define MT_LOG (1<<4)
- #define MT_TALK (1<<5)
- #define MT_CALL (1<<6)
- #define MT_ALL 0xFF
- #define MERROR MT_ERROR, __FILE__, __LINE__
- #define MINFO MT_INFO, __FILE__, __LINE__
- #define MDEBUG MT_DEBUG, __FILE__, __LINE__
- #define MUSER MT_USER, __FILE__, __LINE__
- #define MLOG MT_LOG, __FILE__, __LINE__
- #define MTALK MT_TALK, __FILE__, __LINE__
- #define MCALL MT_CALL, __FILE__, __LINE__

2.6.1 Define Documentation

2.6.1.1 #define ASYNC 1

Definition at line 298 of file midas.h.

Referenced by bm_receive_event(), handFlush(), scan_fragment(), scheduler(), and source_scan().

2.6.1.2 #define EQ_EB (1<<7)

Event run through the event builder

Definition at line 348 of file midas.h.

2.6.1.3 #define EQ_FRAGMENTED (1<<6)

Fragmented Event

Definition at line 347 of file midas.h.

2.6.1.4 #define EQ_INTERRUPT (1<<2)

Interrupt Event

Definition at line 343 of file midas.h.

Referenced by scheduler().

2.6.1.5 #define EQ_MANUAL_TRIG (1<<5)

Manual triggered Event

Definition at line 346 of file midas.h.

2.6.1.6 #define EQ_MULTITHREAD (1<<3)

Multithread Event readout

Definition at line 344 of file midas.h.

Referenced by tr_stop().

2.6.1.7 #define EQ_PERIODIC (1<<0)

Periodic Event

Definition at line 341 of file midas.h.

2.6.1.8 #define EQ_POLLED (1<<1)

Polling Event

Definition at line 342 of file midas.h.

Referenced by register_equipment().

2.6.1.9 #define EQ_SLOW (1<<4)

Slow Control Event

Definition at line 345 of file midas.h.

Referenced by scheduler().

2.6.1.10 #define EVENTID_ALL -1

Definition at line 462 of file midas.h.

Referenced by bm_match_event(), and cm_msg_register().

2.6.1.11 #define FORMAT_ASCII 3

ASCII format

Definition at line 262 of file midas.h.

2.6.1.12 #define FORMAT_DUMP 5

Dump (detailed ASCII) format

Definition at line 264 of file midas.h.

2.6.1.13 #define FORMAT_FIXED 4

Fixed length binary records

Definition at line 263 of file midas.h.

2.6.1.14 #define FORMAT_HBOOK 6

CERN hbook (rz) format

Definition at line 265 of file midas.h.

2.6.1.15 #define FORMAT_MIDAS 1

MIDAS banks

Definition at line 260 of file midas.h.

Referenced by source_scan().

2.6.1.16 #define FORMAT_ROOT 7

CERN ROOT format

Definition at line 266 of file midas.h.

2.6.1.17 #define FORMAT_YBOS 2

YBOS banks

Definition at line 261 of file midas.h.

Referenced by source_scan().

2.6.1.18 #define GET_ALL (1<<0)

get all events (consume)

Definition at line 270 of file midas.h.

Referenced by source_booking().

2.6.1.19 #define GET_FARM (1<<2)

distribute events over several clients (farming)

Definition at line 272 of file midas.h.

2.6.1.20 #define GET_SOME (1<<1)

get as much as possible (sampling)

Definition at line 271 of file midas.h.

Referenced by cm_msg_register().

2.6.1.21 #define MCALL MT_CALL, __FILE__, __LINE__

info message for telephone call

Definition at line 482 of file midas.h.

2.6.1.22 #define MDEBUG MT_DEBUG, __FILE__, __LINE__

-

Definition at line 478 of file midas.h.

Referenced by bm_flush_cache(), bm_push_event(), bm_receive_event(), bm_send_event(), bm_update_read_pointer(), bm_wait_for_free_space(), bm_wakeup_producers(), and cm_transition().

2.6.1.23 #define MERROR MT_ERROR, __FILE__, __LINE__

-

Definition at line 476 of file midas.h.

Referenced by al_check(), al_reset_alarm(), al_trigger_alarm(), analyzer_init(), bm_close_buffer(), bm_copy_from_cache(), bm_flush_cache(), bm_open_buffer(), bm_push_event(), bm_receive_event(), bm_remove_event_request(), bm_request_event(), bm_send_event(), bm_set_cache_size(), bm_skip_event(), bm_validate_client_index(), bm_wait_for_free_space(), cm_check_client(), cm_check_deferred_transition(), cm_cleanup(), cm_connect_experiment1(), cm_deregister_transition(), cm_get_watchdog_info(), cm_list_experiments(), cm_register_deferred_transition(), cm_register_transition(), cm_set_client_info(), cm_set_transition_sequence(), cm_shutdown(), cm_transition(), db_check_record(), db_close_database(), db_copy(), db_copy_xml(), db_create_key(), db_create_link(), db_create_record(), db_delete_key1(), db_enum_key(), db_find_key(), db_get_data(), db_get_data_index(), db_get_key(), db_get_key_info(), db_get_key_time(), db_get_record(), db_get_value(), db_load(), db_lock_database(), db_open_database(), db_open_record(), db_paste(), db_paste_node(), db_protect_database(), db_save(), db_save_struct(), db_save_xml(), db_save_xml_key(), db_set_data(), db_set_data_index(), db_set_record(), db_set_value(), db_unlock_database(), dm_buffer_create(), el_submit(), handFlush(), hs_dump(), load_fragment(), main(), readout_thread(), receive_trigger_event(), register_equipment(), rpc_flush_event(), rpc_register_functions(), rpc_send_event(), rpc_set_option(), scan_fragment(), scheduler(), send_event(), source_booking(), source_scan(), source_unbooking(), tr_start(), tr_stop(), and update_odb().

2.6.1.24 #define MINFO MT_INFO, __FILE__, __LINE__

-

Definition at line 477 of file midas.h.

Referenced by bk_list(), bm_validate_client_pointers(), close_buffers(), cm_check_client(), cm_cleanup(), cm_connect_experiment1(), cm_disconnect_experiment(), cm_set_client_info(), cm_shutdown(), cm_transition(), load_fragment(), register_equipment(), tr_start(), and ybk_list().

2.6.1.25 #define MLOG MT_LOG, __FILE__, __LINE__

info message which is only logged

Definition at line 480 of file midas.h.

2.6.1.26 #define MODE_ALLOC (1<<7)

Definition at line 306 of file midas.h.

Referenced by db_open_record().

2.6.1.27 #define MODE_DELETE (1<<2)

Definition at line 304 of file midas.h.

Referenced by cm_delete_client_info(), cm_deregister_transition(), cm_register_transition(), cm_set_client_info(), and cm_transition().

2.6.1.28 #define MODE_EXCLUSIVE (1<<3)

Definition at line 305 of file midas.h.

Referenced by cm_cleanup(), and db_open_database().

2.6.1.29 #define MODE_READ (1<<0)

Access modes

Definition at line 302 of file midas.h.

Referenced by analyzer_init(), cm_delete_client_info(), cm_deregister_transition(), cm_register_deferred_transition(), cm_register_transition(), cm_set_client_info(), cm_set_transition_sequence(), cm_set_watchdog_params(), cm_transition(), db_create_key(), db_open_database(), db_open_record(), and register_equipment().

2.6.1.30 #define MODE_WRITE (1<<1)

Definition at line 303 of file midas.h.

Referenced by cm_cleanup(), cm_delete_client_info(), cm_deregister_transition(), cm_register_deferred_transition(), cm_register_transition(), cm_set_client_info(), cm_set_transition_sequence(), cm_set_watchdog_params(), cm_transition(), db_create_key(), db_open_database(), and register_equipment().

2.6.1.31 #define MT_ALL 0xFF

-

Definition at line 474 of file midas.h.

Referenced by cm_connect_experiment1(), and main().

2.6.1.32 #define MT_CALL (1<<6)

-

Definition at line 473 of file midas.h.

2.6.1.33 #define MT_DEBUG (1<<2)

-

Definition at line 469 of file midas.h.

2.6.1.34 #define MT_ERROR (1<<0)

-

Definition at line 467 of file midas.h.

2.6.1.35 #define MT_INFO (1<<1)

-

Definition at line 468 of file midas.h.

2.6.1.36 #define MT_LOG (1<<4)

-

Definition at line 471 of file midas.h.

2.6.1.37 #define MT_TALK (1<<5)

-

Definition at line 472 of file midas.h.

2.6.1.38 #define MT_USER (1<<3)

-

Definition at line 470 of file midas.h.

2.6.1.39 #define MTALK MT_TALK, __FILE__, __LINE__

info message for speech system

Definition at line 481 of file midas.h.

Referenced by al_check(), scan_fragment(), and scheduler().

2.6.1.40 #define MUSER MT_USER, __FILE__, __LINE__

produced by interactive user

Definition at line 479 of file midas.h.

2.6.1.41 #define RO_ALWAYS (0xFF)

Always (independent of the run status)

Definition at line 362 of file midas.h.

2.6.1.42 #define RO_BOR (1<<3)

At the Begin of run

Definition at line 356 of file midas.h.

2.6.1.43 #define RO_EOR (1<<4)

At the End of run

Definition at line 357 of file midas.h.

2.6.1.44 #define RO_ODB (1<<8)

Submit data to ODB only

Definition at line 364 of file midas.h.

Referenced by scheduler().

2.6.1.45 #define RO_PAUSE (1<<5)

Before pausing the run

Definition at line 358 of file midas.h.

2.6.1.46 #define RO_PAUSED (1<<2)

???

Definition at line 355 of file midas.h.

2.6.1.47 #define RO_RESUME (1<<6)

Before resuming the run

Definition at line 359 of file midas.h.

2.6.1.48 #define RO_RUNNING (1<<0)

While running

Definition at line 353 of file midas.h.

2.6.1.49 #define RO_STOPPED (1<<1)

Before stopping the run

Definition at line 354 of file midas.h.

2.6.1.50 #define RO_TRANSITIONS (RO_BOR|RO_EOR|RO_PAUSE|RO_RESUME)

At all transitions

Definition at line 361 of file midas.h.

2.6.1.51 #define RPC_CLIENT_HANDLE 9

Definition at line 318 of file midas.h.

Referenced by cm_get_experiment_database(), and cm_set_client_info().

2.6.1.52 #define RPC_CONVERT_FLAGS 7

Definition at line 316 of file midas.h.

Referenced by bm_receive_event(), db_get_record(), db_set_record(), and db_update_record().

2.6.1.53 #define RPC_FTCP 1

Definition at line 324 of file midas.h.

Referenced by cm_transition(), db_send_changed_records(), and scheduler().

2.6.1.54 #define RPC_NODELAY 12

Definition at line 321 of file midas.h.

Referenced by rpc_set_option().

2.6.1.55 #define RPC_OCONVERT_FLAG 3

Definition at line 312 of file midas.h.

2.6.1.56 #define RPC_ODB_HANDLE 8

Definition at line 317 of file midas.h.

Referenced by cm_get_experiment_database(), and cm_set_client_info().

2.6.1.57 #define RPC_OHW_TYPE 4

Definition at line 313 of file midas.h.

Referenced by cm_connect_experiment1().

2.6.1.58 #define RPC_OSERVER_NAME 6

Definition at line 315 of file midas.h.

2.6.1.59 #define RPC_OSERVER_TYPE 5

Definition at line 314 of file midas.h.

Referenced by bm_check_buffers(), bm_close_buffer(), bm_empty_buffers(), bm_open_buffer(), bm_receive_event(), cm_disconnect_experiment(), cm_set_watchdog_params(), db_close_database(), db_get_record(), db_open_database(), and db_set_record().

2.6.1.60 #define RPC_OTIMEOUT 1

RPC options

Definition at line 310 of file midas.h.

Referenced by bm_receive_event(), cm_transition(), main(), and rpc_set_option().

2.6.1.61 #define RPC_OTRANSPORT 2

Definition at line 311 of file midas.h.

Referenced by cm_transition(), db_send_changed_records(), rpc_set_option(), scheduler(), and update_odb().

2.6.1.62 #define RPC_SEND_SOCK 10

Definition at line 319 of file midas.h.

2.6.1.63 #define RPC_TCP 0

Definition at line 323 of file midas.h.

Referenced by cm_transition(), db_send_changed_records(), scheduler(), and update_odb().

2.6.1.64 #define RPC_WATCHDOG_TIMEOUT 11

Definition at line 320 of file midas.h.

Referenced by cm_set_watchdog_params().

2.6.1.65 #define STATE_PAUSED 2

MIDAS run paused

Definition at line 255 of file midas.h.

Referenced by scan_fragment(), and scheduler().

2.6.1.66 #define STATE_RUNNING 3

MIDAS run running

Definition at line 256 of file midas.h.

Referenced by display(), scan_fragment(), and scheduler().

2.6.1.67 #define STATE_STOPPED 1

MIDAS run stopped

Definition at line 254 of file midas.h.

Referenced by display(), scan_fragment(), and scheduler().

2.6.1.68 #define SYNC 0

Synchronous / Asynchronous flags

Definition at line 297 of file midas.h.

Referenced by close_buffers(), cm_check_deferred_transition(), cm_msg(), cm_msg1(), receive_trigger_event(), scheduler(), send_event(), source_scan(), and tr_stop().

2.6.1.69 #define TID_ARRAY 13

array with unknown contents

Definition at line 289 of file midas.h.

2.6.1.70 #define TID_BITFIELD 11

32 Bits Bitfield 0 111... (32)

Definition at line 287 of file midas.h.

Referenced by db_sprintf().

2.6.1.71 #define TID_BOOL 8

four bytes bool 0 1

Definition at line 284 of file midas.h.

Referenced by al_check(), al_trigger_alarm(), ana_end_of_run(), bk_swap(), db_sprintf(), scheduler(), and tr_start().

2.6.1.72 #define TID_BYTE 1

unsigned byte 0 255

Definition at line 277 of file midas.h.

Referenced by db_sprintf().

2.6.1.73 #define TID_CHAR 3

single character 0 255

Definition at line 279 of file midas.h.

Referenced by db_sprintf().

2.6.1.74 #define TID_DOUBLE 10

8 Byte float format

Definition at line 286 of file midas.h.

Referenced by ana_end_of_run(), bk_swap(), db_sprintf(), register_equipment(), and scaler_accum().

2.6.1.75 #define TID_DWORD 6

four bytes 0 2^32-1

Definition at line 282 of file midas.h.

Referenced by bk_swap(), bm_convert_event_header(), bm_open_buffer(), cm_transition(), db_sprintf(), db_update_record(), eb_user(), and read_scaler_event().

2.6.1.76 #define TID_FLOAT 9

4 Byte float format

Definition at line 285 of file midas.h.

Referenced by adc_calib(), bk_swap(), and db_sprintf().

2.6.1.77 #define TID_INT 7

signed dword -2^31 2^31-1

Definition at line 283 of file midas.h.

Referenced by al_check(), al_trigger_alarm(), bk_swap(), cm_connect_client(), cm_connect_experiment1(), cm_delete_client_info(), cm_register_deferred_transition(), cm_register_transition(), cm_set_client_info(), cm_set_transition_sequence(), cm_set_watchdog_params(), cm_shutdown(), cm_transition(), db_sprintf(), el_submit(), load_fragment(), main(), register_equipment(), scheduler(), and tr_start().

2.6.1.78 #define TID_KEY 15

key in online database

Definition at line 291 of file midas.h.

Referenced by db_create_record(), db_delete_key1(), db_paste_node(), and register_equipment().

2.6.1.79 #define TID_LAST 17

end of TID list indicator

Definition at line 293 of file midas.h.

2.6.1.80 #define TID_LINK 16

link in online database

Definition at line 292 of file midas.h.

Referenced by db_create_key(), db_create_link(), db_delete_key1(), db_set_value(), and db_sprintf().

2.6.1.81 #define TID_SBYTE 2

signed byte -128 127

Definition at line 278 of file midas.h.

Referenced by db_sprintf().

2.6.1.82 #define TID_SHORT 5

signed word -32768 32767

Definition at line 281 of file midas.h.

Referenced by bk_swap(), bm_convert_event_header(), and db_sprintf().

2.6.1.83 #define TID_STRING 12

zero terminated string

Definition at line 288 of file midas.h.

Referenced by al_trigger_alarm(), ana_end_of_run(), cm_check_client(), cm_connect_client(), cm_connect_experiment1(), cm_exist(), cm_get_client_info(), cm_msg_log(), cm_msg_log1(), cm_msg_retrieve(), cm_set_client_info(), cm_shutdown(), cm_transition(), db_check_record(), db_copy(), db_create_key(), db_get_value(), db_paste(), db_paste_node(), db_save_xml_key(), db_set_data_index(), db_set_value(), db_sprintf(), el_submit(), load_fragment(), logger_root(), tr_start(), and update_odb().

2.6.1.84 #define TID_STRUCT 14

structure with fixed length

Definition at line 290 of file midas.h.

Referenced by adc_summing(), and bk_close().

2.6.1.85 #define TID_WORD 4

two bytes 0 65535

Definition at line 280 of file midas.h.

Referenced by bk_swap(), db_sprintf(), load_fragment(), read_trigger_event(), and register_equipment().

2.6.1.86 #define TR_DEFERRED (1<<12)

Definition at line 337 of file midas.h.

Referenced by cm_check_deferred_transition().

2.6.1.87 #define TR_PAUSE (1<<2)

Pause transition

Definition at line 335 of file midas.h.

Referenced by cm_deregister_transition(), cm_register_transition(), cm_set_transition_sequence(), cm_transition(), main(), send_all_periodic_events(), and tr_pause().

2.6.1.88 #define TR_RESUME (1<<3)

Resume transition

Definition at line 336 of file midas.h.

Referenced by main(), send_all_periodic_events(), and tr_resume().

2.6.1.89 #define TR_START (1<<0)

Start transition

Definition at line 333 of file midas.h.

Referenced by cm_deregister_transition(), cm_register_transition(), cm_set_transition_sequence(), cm_transition(), main(), scheduler(), send_all_periodic_events(), and tr_start().

2.6.1.90 #define TR_STOP (1<<1)

Stop transition

Definition at line 334 of file midas.h.

Referenced by cm_deregister_transition(), cm_register_transition(), cm_set_transition_sequence(), cm_transition(), main(), scan_fragment(), scheduler(), send_all_periodic_events(), and tr_stop().

2.6.1.91 #define TRIGGER_ALL -1

Definition at line 463 of file midas.h.

Referenced by bm_match_event(), cm_msg_register(), and source_booking().

2.6.1.92 #define WF_CALL_WD (1<<1)

Definition at line 329 of file midas.h.

2.6.1.93 #define WF_WATCH_ME (1<<0)

Watchdog flags

Definition at line 328 of file midas.h.

2.7 Midas Macros

Defines

- #define MAX(a, b) (((a) > (b)) ? (a) : (b))
- #define MIN(a, b) (((a) < (b)) ? (a) : (b))
- #define ALIGN8(x) (((x)+7) & ∼7)
- #define VALIGN(adr, align) (((POINTER_T) (adr)+align-1) & ∼(align-1))

2.7.1 Define Documentation

2.7.1.1 #define ALIGN8(x) (((x)+7) & ∼7)

Align macro for data alignment on 8-byte boundary

Definition at line 447 of file midas.h.

Referenced by bk_close(), bk_find(), bk_iterate(), bk_locate(), bk_swap(), bm_copy_from_cache(), bm_dispatch_from_cache(), bm_flush_cache(), bm_push_event(), bm_receive_event(), bm_send_event(), bm_wait_for_free_space(), db_open_database(), and rpc_send_event().

2.7.1.2 #define MAX(a, b) (((a) > (b)) ? (a) : (b))

MAX

Definition at line 434 of file midas.h.

Referenced by cm_execute(), and scheduler().

2.7.1.3 #define MIN(a, b) (((a) < (b)) ? (a) : (b))

MIN

Definition at line 440 of file midas.h.

Referenced by update_odb().

2.7.1.4 #define VALIGN(adr, align) (((POINTER_T) (adr)+align-1) & ~(align-1))

Align macro for variable data alignment

Definition at line 451 of file midas.h.

Referenced by db_get_record_size(), and update_odb().

2.8 Midas Error definition

Modules

- group Status and error codes
- group Buffer Manager error codes
- group Online Database error codes
- group System Services error code
- group Remote Procedure Calls error codes
- group Other errors

2.9 Midas Structure Declaration

Modules

- group Buffer Section
- group Equipment related
- group Bank related
- group Analyzer related
- group History related
- group ODB runinfo related
- group Alarm related

2.10 Status and error codes

Defines

- #define SUCCESS 1
- #define CM_SUCCESS 1
- #define CM_SET_ERROR 102
- #define CM_NO_CLIENT 103
- #define CM_DB_ERROR 104
- #define CM_UNDEF_EXP 105
- #define CM_VERSION_MISMATCH 106
- #define CM_SHUTDOWN 107
- #define CM_WRONG_PASSWORD 108
- #define CM_UNDEF_ENVIRON 109
- #define CM_DEFERRED_TRANSITION 110
- #define CM_TRANSITION_IN_PROGRESS 111
- #define CM_TIMEOUT 112
- #define CM_INVALID_TRANSITION 113
- #define CM_TOO_MANY_REQUESTS 114

2.10.1 Define Documentation

2.10.1.1 #define CM_DB_ERROR 104

db access error

Definition at line 502 of file midas.h.

2.10.1.2 #define CM_DEFERRED_TRANSITION 110

-

Definition at line 508 of file midas.h.

2.10.1.3 #define CM_INVALID_TRANSITION 113

-

Definition at line 511 of file midas.h.

2.10.1.4 #define CM_NO_CLIENT 103

nobody

Definition at line 501 of file midas.h.

2.10.1.5 #define CM_SET_ERROR 102

set

Definition at line 500 of file midas.h.

2.10.1.6 #define CM_SHUTDOWN 107

-

Definition at line 505 of file midas.h.

2.10.1.7 #define CM_SUCCESS 1

Same

Definition at line 499 of file midas.h.

Referenced by main().

2.10.1.8 #define CM_TIMEOUT 112

-

Definition at line 510 of file midas.h.

2.10.1.9 #define CM_TOO_MANY_REQUESTS 114

-

Definition at line 512 of file midas.h.

2.10.1.10 #define CM_TRANSITION_IN_PROGRESS 111

-

Definition at line 509 of file midas.h.

2.10.1.11 #define CM_UNDEF_ENVIRON 109

-

Definition at line 507 of file midas.h.

2.10.1.12 #define CM_UNDEF_EXP 105

-

Definition at line 503 of file midas.h.

2.10.1.13 #define CM_VERSION_MISMATCH 106

-

Definition at line 504 of file midas.h.

2.10.1.14 #define CM_WRONG_PASSWORD 108

-

Definition at line 506 of file midas.h.

2.10.1.15 #define SUCCESS 1

Success

Definition at line 498 of file midas.h.

Referenced by bm_open_buffer(), cm_transition(), el_submit(), register_equipment(), and scheduler().

2.11 Buffer Manager error codes

Defines

- #define BM_SUCCESS 1
- #define BM_CREATED 202
- #define BM_NO_MEMORY 203
- #define BM_INVALID_NAME 204

- #define BM_INVALID_HANDLE 205
- #define BM_NO_SLOT 206
- #define BM_NO_MUTEX 207
- #define BM_NOT_FOUND 208
- #define BM_ASYNC_RETURN 209
- #define BM_TRUNCATED 210
- #define BM_MULTIPLE_HOSTS 211
- #define BM_MEMSIZE_MISMATCH 212
- #define BM_CONFLICT 213
- #define BM_EXIT 214
- #define BM_INVALID_PARAM 215
- #define BM_MORE_EVENTS 216
- #define BM_INVALID_MIXING 217
- #define BM_NO_SHM 218

2.11.1 Define Documentation

2.11.1.1 #define BM_ASYNC_RETURN 209

-

Definition at line 528 of file midas.h.

Referenced by scan_fragment(), and source_scan().

2.11.1.2 #define BM_CONFLICT 213

-

Definition at line 532 of file midas.h.

2.11.1.3 #define BM_CREATED 202

-

Definition at line 521 of file midas.h.

2.11.1.4 #define BM_EXIT 214

-

Definition at line 533 of file midas.h.

2.11.1.5 #define BM_INVALID_HANDLE 205

-

Definition at line 524 of file midas.h.

2.11.1.6 #define BM_INVALID_MIXING 217

-

Definition at line 536 of file midas.h.

2.11.1.7 #define BM_INVALID_NAME 204

-

Definition at line 523 of file midas.h.

2.11.1.8 #define BM_INVALID_PARAM 215

-

Definition at line 534 of file midas.h.

2.11.1.9 #define BM_MEMSIZE_MISMATCH 212

-

Definition at line 531 of file midas.h.

2.11.1.10 #define BM_MORE_EVENTS 216

-

Definition at line 535 of file midas.h.

2.11.1.11 #define BM_MULTIPLE_HOSTS 211

-

Definition at line 530 of file midas.h.

2.11.1.12 #define BM_NO_MEMORY 203

-

Definition at line 522 of file midas.h.

2.11.1.13 #define BM_NO_MUTEX 207

-

Definition at line 526 of file midas.h.

2.11.1.14 #define BM_NO_SHM 218

-

Definition at line 537 of file midas.h.

2.11.1.15 #define BM_NO_SLOT 206

-

Definition at line 525 of file midas.h.

2.11.1.16 #define BM_NOT_FOUND 208

-

Definition at line 527 of file midas.h.

2.11.1.17 #define BM_SUCCESS 1

-

Definition at line 520 of file midas.h.

Referenced by cm_msg(), cm_msg1(), cm_msg_register(), register_equipment(), and source_scan().

2.11.1.18 #define BM_TRUNCATED 210

-

Definition at line 529 of file midas.h.

2.12 Online Database error codes

Defines

- #define DB_SUCCESS 1
- #define DB_CREATED 302
- #define DB_NO_MEMORY 303
- #define DB_INVALID_NAME 304
- #define DB_INVALID_HANDLE 305
- #define DB_NO_SLOT 306
- #define DB_NO_MUTEX 307
- #define DB_MEMSIZE_MISMATCH 308
- #define DB_INVALID_PARAM 309
- #define DB_FULL 310
- #define DB_KEY_EXIST 311
- #define DB_NO_KEY 312
- #define DB_KEY_CREATED 313
- #define DB_TRUNCATED 314
- #define DB_TYPE_MISMATCH 315
- #define DB_NO_MORE_SUBKEYS 316
- #define DB_FILE_ERROR 317
- #define DB_NO_ACCESS 318
- #define DB_STRUCT_SIZE_MISMATCH 319
- #define DB_OPEN_RECORD 320
- #define DB_OUT_OF_RANGE 321
- #define DB_INVALID_LINK 322
- #define DB_CORRUPTED 323
- #define DB_STRUCT_MISMATCH 324
- #define DB_TIMEOUT 325
- #define DB_VERSION_MISMATCH 326

2.12.1 Define Documentation

2.12.1.1 #define DB_CORRUPTED 323

-

Definition at line 566 of file midas.h.

2.12.1.2 #define DB_CREATED 302

-

Definition at line 545 of file midas.h.

2.12.1.3 #define DB_FILE_ERROR 317

-

Definition at line 560 of file midas.h.

2.12.1.4 #define DB_FULL 310

-

Definition at line 553 of file midas.h.

2.12.1.5 #define DB_INVALID_HANDLE 305

-

Definition at line 548 of file midas.h.

2.12.1.6 #define DB_INVALID_LINK 322

-

Definition at line 565 of file midas.h.

2.12.1.7 #define DB_INVALID_NAME 304

-

Definition at line 547 of file midas.h.

2.12.1.8 #define DB_INVALID_PARAM 309

-

Definition at line 552 of file midas.h.

2.12.1.9 #define DB_KEY_CREATED 313

-

Definition at line 556 of file midas.h.

2.12.1.10 #define DB_KEY_EXIST 311

-

Definition at line 554 of file midas.h.

2.12.1.11 #define DB_MEMSIZE_MISMATCH 308

-

Definition at line 551 of file midas.h.

2.12.1.12 #define DB_NO_ACCESS 318

-

Definition at line 561 of file midas.h.

2.12.1.13 #define DB_NO_KEY 312

-

Definition at line 555 of file midas.h.

Referenced by register_equipment().

2.12.1.14 #define DB_NO_MEMORY 303

-

Definition at line 546 of file midas.h.

2.12.1.15 #define DB_NO_MORE_SUBKEYS 316

-

Definition at line 559 of file midas.h.

2.12.1.16 #define DB_NO_MUTEX 307

-

Definition at line 550 of file midas.h.

2.12.1.17 #define DB_NO_SLOT 306

-

Definition at line 549 of file midas.h.

2.12.1.18 #define DB_OPEN_RECORD 320

-

Definition at line 563 of file midas.h.

2.12.1.19 #define DB_OUT_OF_RANGE 321

-

Definition at line 564 of file midas.h.

2.12.1.20 #define DB_STRUCT_MISMATCH 324

-

Definition at line 567 of file midas.h.

2.12.1.21 #define DB_STRUCT_SIZE_MISMATCH 319

-

Definition at line 562 of file midas.h.

2.12.1.22 #define DB_SUCCESS 1

-

Definition at line 544 of file midas.h.

Referenced by al_check(), al_trigger_alarm(), cm_connect_experiment1(), cm_shutdown(), cm_transition(), db_check_record(), db_delete_key1(), db_paste_node(), readout_thread(), and tr_start().

2.12.1.23 #define DB_TIMEOUT 325

-

Definition at line 568 of file midas.h.

2.12.1.24 #define DB_TRUNCATED 314

-

Definition at line 557 of file midas.h.

2.12.1.25 #define DB_TYPE_MISMATCH 315

-

Definition at line 558 of file midas.h.

2.12.1.26 #define DB_VERSION_MISMATCH 326

-

Definition at line 569 of file midas.h.

2.13 System Services error code

Defines

- #define SS_SUCCESS 1
- #define SS_CREATED 402
- #define SS_NO_MEMORY 403
- #define SS_INVALID_NAME 404
- #define SS_INVALID_HANDLE 405
- #define SS_INVALID_ADDRESS 406
- #define SS_FILE_ERROR 407
- #define SS_NO_MUTEX 408
- #define SS_NO_PROCESS 409
- #define SS_NO_THREAD 410
- #define SS_SOCKET_ERROR 411
- #define SS_TIMEOUT 412
- #define SS_SERVER_RECV 413
- #define SS_CLIENT_RECV 414
- #define SS_ABORT 415
- #define SS_EXIT 416
- #define SS_NO_TAPE 417
- #define SS_DEV_BUSY 418
- #define SS_IO_ERROR 419
- #define SS_TAPE_ERROR 420
- #define SS_NO_DRIVER 421
- #define SS_END_OF_TAPE 422
- #define SS_END_OF_FILE 423
- #define SS_FILE_EXISTS 424
- #define SS_NO_SPACE 425
- #define SS_INVALID_FORMAT 426
- #define SS_NO_ROOT 427
- #define SS_SIZE_MISMATCH 428

2.13.1 Define Documentation

2.13.1.1 #define SS_ABORT 415

-

Definition at line 590 of file midas.h.

2.13.1.2 #define SS_CLIENT_RECV 414

-

Definition at line 589 of file midas.h.

2.13.1.3 #define SS_CREATED 402

-

Definition at line 577 of file midas.h.

Referenced by bm_open_buffer(), cm_connect_experiment1(), device_driver(), and dm_buffer_create().

2.13.1.4 #define SS_DEV_BUSY 418

-

Definition at line 593 of file midas.h.

2.13.1.5 #define SS_END_OF_FILE 423

-

Definition at line 598 of file midas.h.

2.13.1.6 #define SS_END_OF_TAPE 422

-

Definition at line 597 of file midas.h.

2.13.1.7 #define SS_EXIT 416

-

Definition at line 591 of file midas.h.

2.13.1.8 #define SS_FILE_ERROR 407

-

Definition at line 582 of file midas.h.

2.13.1.9 #define SS_FILE_EXISTS 424

-

Definition at line 599 of file midas.h.

2.13.1.10 #define SS_INVALID_ADDRESS 406

-

Definition at line 581 of file midas.h.

2.13.1.11 #define SS_INVALID_FORMAT 426

-

Definition at line 601 of file midas.h.

2.13.1.12 #define SS_INVALID_HANDLE 405

-

Definition at line 580 of file midas.h.

2.13.1.13 #define SS_INVALID_NAME 404

-

Definition at line 579 of file midas.h.

2.13.1.14 #define SS_IO_ERROR 419

-

Definition at line 594 of file midas.h.

2.13.1.15 #define SS_NO_DRIVER 421

-

Definition at line 596 of file midas.h.

2.13.1.16 #define SS_NO_MEMORY 403

-

Definition at line 578 of file midas.h.

Referenced by db_open_database().

2.13.1.17 #define SS_NO_MUTEX 408

-

Definition at line 583 of file midas.h.

2.13.1.18 #define SS_NO_PROCESS 409

-

Definition at line 584 of file midas.h.

2.13.1.19 #define SS_NO_ROOT 427

-

Definition at line 602 of file midas.h.

2.13.1.20 #define SS_NO_SPACE 425

-

Definition at line 600 of file midas.h.

2.13.1.21 #define SS_NO_TAPE 417

-

Definition at line 592 of file midas.h.

2.13.1.22 #define SS_NO_THREAD 410

-

Definition at line 585 of file midas.h.

2.13.1.23 #define SS_SERVER_RECV 413

-

Definition at line 588 of file midas.h.

2.13.1.24 #define SS_SIZE_MISMATCH 428

-

Definition at line 603 of file midas.h.

2.13.1.25 #define SS_SOCKET_ERROR 411

-

Definition at line 586 of file midas.h.

2.13.1.26 #define SS_SUCCESS 1

-

Definition at line 576 of file midas.h.

Referenced by bm_open_buffer(), db_open_database(), and ss_thread_kill().

2.13.1.27 #define SS_TAPE_ERROR 420

-

Definition at line 595 of file midas.h.

2.13.1.28 #define SS_TIMEOUT 412

-

Definition at line 587 of file midas.h.

2.14 Remote Procedure Calls error codes

Defines

- #define RPC_SUCCESS 1
- #define RPC_ABORT SS_ABORT
- #define RPC_NO_CONNECTION 502
- #define RPC_NET_ERROR 503
- #define RPC_TIMEOUT 504
- #define RPC_EXCEED_BUFFER 505
- #define RPC_NOT_REGISTERED 506
- #define RPC_CONNCLOSED 507
- #define RPC_INVALID_ID 508
- #define RPC_SHUTDOWN 509
- #define RPC_NO_MEMORY 510
- #define RPC_DOUBLE_DEFINED 511
- #define RPC_MUTEX_TIMEOUT 512

2.14.1 Define Documentation

2.14.1.1 #define RPC_ABORT SS_ABORT

-

Definition at line 611 of file midas.h.

2.14.1.2 #define RPC_CONNCLOSED 507

-

Definition at line 617 of file midas.h.

2.14.1.3 #define RPC_DOUBLE_DEFINED 511

-

Definition at line 621 of file midas.h.

2.14.1.4 #define RPC_EXCEED_BUFFER 505

-

Definition at line 615 of file midas.h.

2.14.1.5 #define RPC_INVALID_ID 508

-

Definition at line 618 of file midas.h.

2.14.1.6 #define RPC_MUTEX_TIMEOUT 512

-

Definition at line 622 of file midas.h.

2.14.1.7 #define RPC_NET_ERROR 503

-

Definition at line 613 of file midas.h.

2.14.1.8 #define RPC_NO_CONNECTION 502

-

Definition at line 612 of file midas.h.

2.14.1.9 #define RPC_NO_MEMORY 510

-

Definition at line 620 of file midas.h.

2.14.1.10 #define RPC_NOT_REGISTERED 506

-

Definition at line 616 of file midas.h.

2.14.1.11 #define RPC_SHUTDOWN 509

-

Definition at line 619 of file midas.h.

Referenced by scan_fragment(), and scheduler().

2.14.1.12 #define RPC_SUCCESS 1

-

Definition at line 610 of file midas.h.

2.14.1.13 #define RPC_TIMEOUT 504

-

Definition at line 614 of file midas.h.

2.15 Other errors

Defines

- #define FE_SUCCESS 1
- #define FE_ERR_ODB 602
- #define FE_ERR_HW 603
- #define FE_ERR_DISABLED 604

- #define FE_ERR_DRIVER 605
- #define HS_SUCCESS 1
- #define HS_FILE_ERROR 702
- #define HS_NO_MEMORY 703
- #define HS_TRUNCATED 704
- #define HS_WRONG_INDEX 705
- #define HS_UNDEFINED_EVENT 706
- #define HS_UNDEFINED_VAR 707
- #define FTP_SUCCESS 1
- #define FTP_NET_ERROR 802
- #define FTP_FILE_ERROR 803
- #define FTP_RESPONSE_ERROR 804
- #define FTP_INVALID_ARG 805
- #define EL_SUCCESS 1
- #define EL_FILE_ERROR 902
- #define EL_NO_MESSAGE 903
- #define EL_TRUNCATED 904
- #define EL_FIRST_MSG 905
- #define EL_LAST_MSG 906
- #define AL_SUCCESS 1
- #define AL_INVALID_NAME 1002
- #define AL_ERROR_ODB 1003
- #define AL_RESET 1004
- #define CMD_INIT 1
- #define CMD_WRITE 100
- #define CMD_INTERRUPT_ENABLE 100
- #define BD_GETS(s, z, p, t) info → bd(CMD_GETS, info → bd_info, s, z, p, t)

2.15.1 Define Documentation

2.15.1.1 #define AL_ERROR_ODB 1003

-

Definition at line 666 of file midas.h.

2.15.1.2 #define AL_INVALID_NAME 1002

-

Definition at line 665 of file midas.h.

2.15.1.3 #define AL_RESET 1004

-

Definition at line 667 of file midas.h.

2.15.1.4 #define AL_SUCCESS 1

-

Definition at line 664 of file midas.h.

2.15.1.5 #define BD_GETS(s, z, p, t) info → bd(CMD_GETS, info → bd_info, s, z, p, t)

macros for bus driver access

Definition at line 727 of file midas.h.

2.15.1.6 #define BD_PUTS(s) info → bd(CMD_PUTS, info → bd_info, s)

Definition at line 729 of file midas.h.

2.15.1.7 #define BD_READS(s, z, p, t) info → bd(CMD_READ, info → bd_info, s, z, p, t)

Definition at line 728 of file midas.h.

2.15.1.8 #define BD_WRITES(s) info → bd(CMD_WRITE, info → bd_info, s)

Definition at line 730 of file midas.h.

2.15.1.9 #define CMD_DEBUG 104

Definition at line 715 of file midas.h.

2.15.1.10 #define CMD_DISABLE_COMMAND (1<<15)

Definition at line 707 of file midas.h.

2.15.1.11 #define CMD_ENABLE_COMMAND (1<<14)

Definition at line 706 of file midas.h.

2.15.1.12 #define CMD_EXIT 2

Definition at line 672 of file midas.h.

Referenced by device_driver(), and main().

2.15.1.13 #define CMD_GET CMD_GET_FIRST

Definition at line 693 of file midas.h.

2.15.1.14 #define CMD_GET_CURRENT CMD_GET_FIRST+1

Definition at line 694 of file midas.h.

2.15.1.15 #define CMD_GET_CURRENT_LIMIT CMD_GET_DIRECT+2

Definition at line 700 of file midas.h.

2.15.1.16 #define CMD_GET_DEMAND CMD_GET_DIRECT

Definition at line 698 of file midas.h.

Referenced by device_driver().

2.15.1.17 #define CMD_GET_DIRECT CMD_GET_LAST+1

Definition at line 697 of file midas.h.

2.15.1.18 #define CMD_GET_DIRECT_LAST CMD_GET_DIRECT+5

Definition at line 704 of file midas.h.

2.15.1.19 #define CMD_GET_FIRST CMD_SET_LAST+1

Definition at line 692 of file midas.h.

Referenced by device_driver().

2.15.1.20 #define CMD_GET_LABEL 10

Definition at line 680 of file midas.h.

Referenced by device_driver().

2.15.1.21 #define CMD_GET_LAST CMD_GET_FIRST+1

Definition at line 695 of file midas.h.

2.15.1.22 #define CMD_GET_RAMPDOWN CMD_GET_DIRECT+4

Definition at line 702 of file midas.h.

2.15.1.23 #define CMD_GET_RAMPUP CMD_GET_DIRECT+3

Definition at line 701 of file midas.h.

2.15.1.24 #define CMD_GET_THRESHOLD 6

Definition at line 676 of file midas.h.

2.15.1.25 #define CMD_GET_THRESHOLD_CURRENT 7

Definition at line 677 of file midas.h.

2.15.1.26 #define CMD_GET_THRESHOLD_ZERO 8

Definition at line 678 of file midas.h.

2.15.1.27 #define CMD_GET_TRIP_TIME CMD_GET_DIRECT+5

Definition at line 703 of file midas.h.

2.15.1.28 #define CMD_GET_VOLTAGE_LIMIT CMD_GET_DIRECT+1

Definition at line 699 of file midas.h.

2.15.1.29 #define CMD_GETS 103

Definition at line 714 of file midas.h.

2.15.1.30 #define CMD_IDLE 5

Definition at line 675 of file midas.h.

Referenced by scheduler().

2.15.1.31 #define CMD_INIT 1

Slow control device driver commands

Definition at line 671 of file midas.h.

Referenced by device_driver(), and register_equipment().

2.15.1.32 #define CMD_INTERRUPT_ATTACH 102

Definition at line 722 of file midas.h.

Referenced by interrupt_configure(), and register_equipment().

2.15.1.33 #define CMD_INTERRUPT_DETACH 103

Definition at line 723 of file midas.h.

Referenced by interrupt_configure(), and main().

2.15.1.34 #define CMD_INTERRUPT_DISABLE 101

Definition at line 721 of file midas.h.

Referenced by interrupt_configure(), main(), and readout_enable().

2.15.1.35 #define CMD_INTERRUPT_ENABLE 100

Commands for interrupt events

Definition at line 720 of file midas.h.

Referenced by interrupt_configure(), and readout_enable().

2.15.1.36 #define CMD_MISC_LAST 10

Definition at line 681 of file midas.h.

2.15.1.37 #define CMD_NAME 105

Definition at line 716 of file midas.h.

2.15.1.38 #define CMD_PUTS 102

Definition at line 713 of file midas.h.

2.15.1.39 #define CMD_READ 101

Definition at line 712 of file midas.h.

2.15.1.40 #define CMD_SET CMD_SET_FIRST

Definition at line 684 of file midas.h.

2.15.1.41 #define CMD_SET_CURRENT_LIMIT CMD_SET_FIRST+2

Definition at line 686 of file midas.h.

2.15.1.42 #define CMD_SET_FIRST CMD_MISC_LAST+1

Definition at line 683 of file midas.h.

Referenced by device_driver().

2.15.1.43 #define CMD_SET_LABEL 9

Definition at line 679 of file midas.h.

Referenced by device_driver().

2.15.1.44 #define CMD_SET_LAST CMD_SET_FIRST+5

Definition at line 690 of file midas.h.

2.15.1.45 #define CMD_SET_RAMPDOWN CMD_SET_FIRST+4

Definition at line 688 of file midas.h.

2.15.1.46 #define CMD_SET_RAMPUP CMD_SET_FIRST+3

Definition at line 687 of file midas.h.

2.15.1.47 #define CMD_SET_TRIP_TIME CMD_SET_FIRST+5

Definition at line 689 of file midas.h.

2.15.1.48 #define CMD_SET_VOLTAGE_LIMIT CMD_SET_FIRST+1

Definition at line 685 of file midas.h.

2.15.1.49 #define CMD_START 3

Definition at line 673 of file midas.h.

Referenced by device_driver(), and register_equipment().

2.15.1.50 #define CMD_STOP 4

Definition at line 674 of file midas.h.

Referenced by device_driver(), and main().

2.15.1.51 #define CMD_WRITE 100

Slow control bus driver commands

Definition at line 711 of file midas.h.

2.15.1.52 #define EL_FILE_ERROR 902

-

Definition at line 656 of file midas.h.

2.15.1.53 #define EL_FIRST_MSG 905

-

Definition at line 659 of file midas.h.

2.15.1.54 #define EL_LAST_MSG 906

-

Definition at line 660 of file midas.h.

2.15.1.55 #define EL_NO_MESSAGE 903

-

Definition at line 657 of file midas.h.

2.15.1.56 #define EL_SUCCESS 1

-

Definition at line 655 of file midas.h.

2.15.1.57 #define EL_TRUNCATED 904

-

Definition at line 658 of file midas.h.

2.15.1.58 #define FE_ERR_DISABLED 604

-

Definition at line 632 of file midas.h.

2.15.1.59 #define FE_ERR_DRIVER 605

-

Definition at line 633 of file midas.h.

2.15.1.60 #define FE_ERR_HW 603

-

Definition at line 631 of file midas.h.

2.15.1.61 #define FE_ERR_ODB 602

-

Definition at line 630 of file midas.h.

2.15.1.62 #define FE_SUCCESS 1

-

Definition at line 629 of file midas.h.

Referenced by device_driver(), and register_equipment().

2.15.1.63 #define FTP_FILE_ERROR 803

-

Definition at line 649 of file midas.h.

2.15.1.64 #define FTP_INVALID_ARG 805

-

Definition at line 651 of file midas.h.

2.15.1.65 #define FTP_NET_ERROR 802

-

Definition at line 648 of file midas.h.

2.15.1.66 #define FTP_RESPONSE_ERROR 804

-

Definition at line 650 of file midas.h.

2.15.1.67 #define FTP_SUCCESS 1

-

Definition at line 647 of file midas.h.

2.15.1.68 #define HS_FILE_ERROR 702

-

Definition at line 638 of file midas.h.

2.15.1.69 #define HS_NO_MEMORY 703

-

Definition at line 639 of file midas.h.

2.15.1.70 #define HS_SUCCESS 1

-

Definition at line 637 of file midas.h.

2.15.1.71 #define HS_TRUNCATED 704

-

Definition at line 640 of file midas.h.

2.15.1.72 #define HS_UNDEFINED_EVENT 706

-

Definition at line 642 of file midas.h.

2.15.1.73 #define HS_UNDEFINED_VAR 707

-

Definition at line 643 of file midas.h.

2.15.1.74 #define HS_WRONG_INDEX 705

-

Definition at line 641 of file midas.h.

2.16 Buffer Section

Data Structures

- struct EVENT_HEADER
- struct EVENT_REQUEST
- struct BUFFER_CLIENT
- struct BUFFER_HEADER
- struct BUFFER
- struct KEY
- struct KEYLIST

Defines

- #define TRIGGER_MASK(e) ((((EVENT_HEADER ∗) e)-1) → trigger_mask)
- #define EVENT_ID(e) ((((EVENT_HEADER ∗) e)-1) → event_id)
- #define SERIAL_NUMBER(e) ((((EVENT_HEADER ∗) e)-1) → serial_number)
- #define TIME_STAMP(e) ((((EVENT_HEADER ∗) e)-1) → time_stamp)
- #define EVENTID_BOR ((short int) 0x8000)
- #define EVENTID_EOR ((short int) 0x8001)
- #define EVENTID_MESSAGE ((short int) 0x8002)
- #define EVENTID_FRAG1 ((unsigned short) 0xC000)
- #define MIDAS_MAGIC 0x494d

2.16.1 Define Documentation

2.16.1.1 #define EVENT_ID(e) ((((EVENT_HEADER ∗) e)-1) → event_id)

EVENT_ID Extract or set the event ID field pointed by the argument..

Parameters:

e pointer to the midas event (pevent)

Definition at line 775 of file midas.h.

2.16.1.2 #define EVENT_SOURCE(e, o) (∗ (INT∗) (e+o))

Definition at line 790 of file midas.h.

2.16.1.3 #define EVENTID_BOR ((short int) 0x8000)

Begin-of-run

Definition at line 794 of file midas.h.

2.16.1.4 #define EVENTID_EOR ((short int) 0x8001)

End-of-run

Definition at line 795 of file midas.h.

2.16.1.5 #define EVENTID_FRAG ((unsigned short) 0xD000)

Definition at line 801 of file midas.h.

2.16.1.6 #define EVENTID_FRAG1 ((unsigned short) 0xC000)

fragmented events

Definition at line 800 of file midas.h.

Referenced by bm_dispatch_event(), and bm_match_event().

2.16.1.7 #define EVENTID_MESSAGE ((short int) 0x8002)

Message events

Definition at line 796 of file midas.h.

Referenced by cm_msg(), and cm_msg1().

2.16.1.8 #define MIDAS_MAGIC 0x494d

'MI'

Definition at line 805 of file midas.h.

2.16.1.9 #define SERIAL_NUMBER(e) ((((EVENT_HEADER ∗) e)-1) → serial_number)

SERIAL_NUMBER Extract or set/reset the serial number field pointed by the argument.

Parameters:

- ***e*** pointer to the midas event (pevent)

Definition at line 782 of file midas.h.

2.16.1.10 #define TIME_STAMP(e) ((((EVENT_HEADER ∗) e)-1) → time_stamp)

TIME_STAMP Extract or set/reset the time stamp field pointed by the argument.

Parameters:

- ***e*** pointer to the midas event (pevent)

Definition at line 789 of file midas.h.

2.16.1.11 #define TRIGGER_MASK(e) ((((EVENT_HEADER ∗) e)-1) → trigger_mask)

TRIGGER_MASK Extract or set the trigger mask field pointed by the argument.

Parameters:

e pointer to the midas event (pevent)

Definition at line 768 of file midas.h.

2.17 Equipment related

Data Structures

- struct BUS_DRIVER
- struct DD_MT_CHANNEL
- struct DD_MT_BUFFER
- struct DEVICE_DRIVER
- struct EQUIPMENT_INFO
- struct EQUIPMENT_STATS
- struct eqpmnt

Defines

- #define DF_INPUT (1<<0)
- #define DF_OUTPUT (1<<1)
- #define DF_PRIO_DEVICE (1<<2)
- #define DF_READ_ONLY (1<<3)

2.17.1 Define Documentation

2.17.1.1 #define DF_HW_RAMP (1<<5)

Definition at line 913 of file midas.h.

2.17.1.2 #define DF_INPUT (1<<0)

channel is input

Definition at line 908 of file midas.h.

2.17.1.3 #define DF_MULTITHREAD (1<<4)

Definition at line 912 of file midas.h.

Referenced by device_driver().

2.17.1.4 #define DF_OUTPUT (1<<1)

channel is output

Definition at line 909 of file midas.h.

2.17.1.5 #define DF_PRIO_DEVICE (1<<2)

get demand values from device instead of ODB

Definition at line 910 of file midas.h.

2.17.1.6 #define DF_READ_ONLY (1<<3)

never write demand values to device

Definition at line 911 of file midas.h.

2.17.2 Typedef Documentation

2.17.2.1 typedef struct eqpmnt EQUIPMENT

Referenced by close_buffers(), receive_trigger_event(), scan_fragment(), scheduler(), send_event(), and tr_stop().

2.17.2.2 typedef struct eqpmnt* PEQUIPMENT

Definition at line 972 of file midas.h.

2.17.3 Function Documentation

2.17.3.1 INT device_driver (DEVICE_DRIVER * *device_driver*, INT *cmd*, ...)

Definition at line 375 of file mfe.c.

2.18 Bank related

Data Structures

- struct BANK_HEADER
- struct BANK

- struct BANK32
- struct TAG
- struct BANK_LIST

Defines

- #define BANK_FORMAT_VERSION 1
- #define BANK_FORMAT_32BIT (1<<4)

2.18.1 Define Documentation

2.18.1.1 #define BANK_FORMAT_32BIT (1<<4)

-

Definition at line 1007 of file midas.h.

2.18.1.2 #define BANK_FORMAT_VERSION 1

-

Definition at line 1006 of file midas.h.

Referenced by bk_init32().

2.19 Analyzer related

Data Structures

- struct ANA_MODULE
- struct AR_INFO
- struct AR_STATS
- struct ANALYZE_REQUEST
- struct ANA_OUTPUT_INFO
- struct ANA_TEST

2.19.1 Define Documentation

2.19.1.1 #define ANA_OUTPUT_INFO_STR "\Filename = STRING : [256] run%05d.asc\n\RWNT = BOOL : 0\n\Histo Dump = BOOL : 0\n\Histo Dump Filename = STRING : [256] his%05d.rz\n\Clear histos = BOOL : 1\n\Last Histo Filename = STRING : [256] last.rz\n\Events to ODB = BOOL : 1\n\Global Memory Name = STRING : [8] ONLN\n\"

Definition at line 1120 of file midas.h.

2.19.1.2 #define DEF_TEST(t) extern ANA_TEST t;

Definition at line 1147 of file midas.h.

2.19.1.3 #define SET_TEST(t, v) { if (!t.registered) test_register(&t); t.value = (v); }

Definition at line 1141 of file midas.h.

Referenced by adc_summing().

2.19.1.4 #define TEST(t) (t.value)

Definition at line 1142 of file midas.h.

2.20 History related

Data Structures

- struct HIST_RECORD
- struct DEF_RECORD
- struct INDEX_RECORD
- struct HISTORY

2.20.1 Define Documentation

2.20.1.1 #define RT_DATA (∗((DWORD ∗) "HSDA"))

Definition at line 1158 of file midas.h.

2.20.1.2 #define RT_DEF (∗((DWORD ∗) "HSDF"))

Definition at line 1159 of file midas.h.

2.21 ODB runinfo related

Data Structures

- struct RUNINFO

2.21.1 Define Documentation

2.21.1.1 #define RUNINFO_STR(_name)

Value:

```
char *_name[] = {\
"[.]",\
"State = INT : 1",\
"Online Mode = INT : 1",\
"Run number = INT : 0",\
"Transition in progress = INT : 0",\
"Requested transition = INT : 0",\
"Start time = STRING : [32] Tue Sep 09 15:04:42 1997",\
"Start time binary = DWORD : 0",\
"Stop time = STRING : [32] Tue Sep 09 15:04:42 1997",\
"Stop time binary = DWORD : 0",\
"",\
NULL }
```

Definition at line 1216 of file midas.h.

Referenced by analyzer_init(), and cm_connect_experiment1().

2.22 Alarm related

2.22.1 Detailed Description

Alarm structure.

Data Structures

- struct PROGRAM_INFO
- struct ALARM_CLASS
- struct ALARM

Defines

- #define AT_INTERNAL 1
- #define AT_PROGRAM 2
- #define AT_EVALUATED 3
- #define AT_PERIODIC 4
- #define AT_LAST 4

2.22.2 Define Documentation

2.22.2.1 #define ALARM_CLASS_STR(_name)

Value:

```
char *_name[] = {\
"[.]",\
"Write system message = BOOL : y",\
"Write Elog message = BOOL : n",\
"System message interval = INT : 60",\
"System message last = DWORD : 0",\
"Execute command = STRING : [256] ",\
"Execute interval = INT : 0",\
"Execute last = DWORD : 0",\
"Stop run = BOOL : n",\
"Display BGColor = STRING : [32] red",\
"Display FGColor = STRING : [32] black",\
"",\
NULL }
```

Definition at line 1288 of file midas.h.

Referenced by al_check().

2.22.2.2 #define ALARM_ODB_STR(_name)

Value:

```
char *_name[] = {\
"[.]",\
"Active = BOOL : n",\
"Triggered = INT : 0",\
"Type = INT : 3",\
"Check interval = INT : 60",\
"Checked last = DWORD : 0",\
"Time triggered first = STRING : [32] ",\
"Time triggered last = STRING : [32] ",\
"Condition = STRING : [256] /Runinfo/Run number > 100",\
"Alarm Class = STRING : [32] Alarm",\
"Alarm Message = STRING : [80] Run number became too large",\
"",\
NULL }
```

Definition at line 1318 of file midas.h.

Referenced by al_check(), and al_trigger_alarm().

2.22.2.3 #define ALARM_PERIODIC_STR(_name)

Value:

```
char *_name[] = {\
"[.]",\
"Active = BOOL : n",\
"Triggered = INT : 0",\
"Type = INT : 4",\
"Check interval = INT : 28800",\
"Checked last = DWORD : 0",\
"Time triggered first = STRING : [32] ",\
"Time triggered last = STRING : [32] ",\
"Condition = STRING : [256] ",\
"Alarm Class = STRING : [32] Warning",\
"Alarm Message = STRING : [80] Please do your shift checks",\
"",\
NULL }
```

Definition at line 1333 of file midas.h.

Referenced by al_check().

2.22.2.4 #define AT_EVALUATED 3

-

Definition at line 1255 of file midas.h.

Referenced by al_check(), and al_trigger_alarm().

2.22.2.5 #define AT_INTERNAL 1

-

Definition at line 1253 of file midas.h.

2.22.2.6 #define AT_LAST 4

-

Definition at line 1257 of file midas.h.

2.22.2.7 #define AT_PERIODIC 4

-

Definition at line 1256 of file midas.h.

Referenced by al_check().

2.22.2.8 #define AT_PROGRAM 2

-

Definition at line 1254 of file midas.h.

Referenced by al_check().

2.22.2.9 #define PROGRAM_INFO_STR(_name)

Value:

```
char *_name[] = {\
"[.]",\
"Required = BOOL : n",\
"Watchdog timeout = INT : 10000",\
"Check interval = DWORD : 180000",\
"Start command = STRING : [256] ",\
"Auto start = BOOL : n",\
"Auto stop = BOOL : n",\
"Auto restart = BOOL : n",\
"Alarm class = STRING : [32] ",\
"First failed = DWORD : 0",\
"",\
NULL }
```

Definition at line 1259 of file midas.h.

Referenced by cm_set_client_info().

2.23 The ybos.h & ybos.c

Modules

- group YBOS Define
- group YBOS error code
- group YBOS Macros
- group YBOS Bank Functions (ybk_xxx)

2.24 YBOS Define

Defines

- #define YBOS_PHYREC_SIZE 8192
- #define YBOS_BUFFER_SIZE 3∗(YBOS_PHYREC_SIZE<<2) + MAX_EVENT_SIZE + 128
- #define YB_BANKLIST_MAX 32
- #define YB_STRING_BANKLIST_MAX YB_BANKLIST_MAX ∗ 4
- #define H_BLOCK_SIZE 0
- #define H_BLOCK_NUM 1
- #define H_HEAD_LEN 2
- #define H_START 3
- #define D_RECORD 1
- #define D_HEADER 2
- #define D_EVTLEN 3
- #define YB_COMPLETE 1
- #define YB_INCOMPLETE 2
- #define YB_NO_RECOVER -1
- #define YB_NO_RUN 0
- #define YB_ADD_RUN 1
- #define DSP_RAW 1
- #define DSP_RAW_SINGLE 2
- #define DSP_BANK 3
- #define DSP_BANK_SINGLE 4
- #define DSP_UNK 0
- #define DSP_DEC 1
- #define DSP_HEX 2
- #define DSP_ASC 3
- #define I2_BKTYPE 1
- #define A1_BKTYPE 2
- #define I4_BKTYPE 3

- #define F4_BKTYPE 4
- #define D8_BKTYPE 5
- #define I1_BKTYPE 8
- #define MAX_BKTYPE I1_BKTYPE+1

2.24.1 Define Documentation

2.24.1.1 #define A1_BKTYPE 2

ASCII 1 byte

Definition at line 278 of file ybos.h.

2.24.1.2 #define D8_BKTYPE 5

Double 8 bytes

Definition at line 281 of file ybos.h.

2.24.1.3 #define D_EVTLEN 3

YBOS

Definition at line 109 of file ybos.h.

2.24.1.4 #define D_HEADER 2

YBOS

Definition at line 108 of file ybos.h.

2.24.1.5 #define D_RECORD 1

YBOS

Definition at line 107 of file ybos.h.

2.24.1.6 #define DSP_ASC 3

Display data in ASCII format

Definition at line 131 of file ybos.h.

2.24.1.7 #define DSP_BANK 3

Display data in bank format

Definition at line 123 of file ybos.h.

2.24.1.8 #define DSP_BANK_SINGLE 4

Display only requested data in bank format

Definition at line 124 of file ybos.h.

2.24.1.9 #define DSP_DEC 1

Display data in decimal format

Definition at line 129 of file ybos.h.

2.24.1.10 #define DSP_HEX 2

Display data in headecimal format

Definition at line 130 of file ybos.h.

2.24.1.11 #define DSP_RAW 1

Display raw data

Definition at line 121 of file ybos.h.

2.24.1.12 #define DSP_RAW_SINGLE 2

Display raw data (no single mode)

Definition at line 122 of file ybos.h.

2.24.1.13 #define DSP_UNK 0

Display format unknown

Definition at line 128 of file ybos.h.

2.24.1.14 #define F4_BKTYPE 4

Float 4 bytes

Definition at line 280 of file ybos.h.

2.24.1.15 #define H_BLOCK_NUM 1

YBOS

Definition at line 101 of file ybos.h.

2.24.1.16 #define H_BLOCK_SIZE 0

YBOS

Definition at line 100 of file ybos.h.

2.24.1.17 #define H_HEAD_LEN 2

YBOS

Definition at line 102 of file ybos.h.

2.24.1.18 #define H_START 3

YBOS

Definition at line 103 of file ybos.h.

2.24.1.19 #define I1_BKTYPE 8

Signed Integer 1 byte

Definition at line 282 of file ybos.h.

Referenced by update_odb().

2.24.1.20 #define I2_BKTYPE 1

Signed Integer 2 bytes

Definition at line 277 of file ybos.h.

2.24.1.21 #define I4_BKTYPE 3

Signed Interger 4bytes

Definition at line 279 of file ybos.h.

2.24.1.22 #define MAX_BKTYPE I1_BKTYPE+1

delimiter

Definition at line 283 of file ybos.h.

2.24.1.23 #define YB_ADD_RUN 1

YBOS

Definition at line 117 of file ybos.h.

2.24.1.24 #define YB_BANKLIST_MAX 32

maximum number of banks to be found by the ybk_list() or bk_list()

Definition at line 69 of file ybos.h.

Referenced by ybk_list().

2.24.1.25 #define YB_COMPLETE 1

YBOS

Definition at line 113 of file ybos.h.

2.24.1.26 #define YB_INCOMPLETE 2

YBOS

Definition at line 114 of file ybos.h.

2.24.1.27 #define YB_NO_RECOVER -1

YBOS

Definition at line 115 of file ybos.h.

2.24.1.28 #define YB_NO_RUN 0

YBOS

Definition at line 116 of file ybos.h.

2.24.1.29 #define YB_STRING_BANKLIST_MAX YB_BANKLIST_MAX $*$ 4

to be used for xbk_list()

Definition at line 71 of file ybos.h.

2.24.1.30 #define YBOS_BUFFER_SIZE 3$*$(YBOS_PHYREC_SIZE<<2) + MAX_EVENT_SIZE + 128

in BYTES

Definition at line 67 of file ybos.h.

2.24.1.31 #define YBOS_HEADER_LENGTH 4

Definition at line 66 of file ybos.h.

2.24.1.32 #define YBOS_PHYREC_SIZE 8192

I∗4

Definition at line 64 of file ybos.h.

2.25 YBOS Macros

Defines

- #define SWAP_D2WORD(_d2w)
- #define EVID_TRINAT
- #define YBOS_EVID_BANK(__a, __b, __c,__d,__e)
- #define MIDAS_EVID_BANK(__a, __b, __c,__d,__e)

2.25.1 Define Documentation

2.25.1.1 #define EVID_TRINAT

As soon as the Midas header is striped out from the event, the YBOS remaining data has lost the event synchonization unless included by the user. It is therefore necessary to have a YBOS bank duplicating this information usually done in the FE by creating a "EVID" bank filled with the Midas info and other user information.

Unfortunately the format of this EVID is flexible and I couldn't force user to use a default structure. For this reason, I'm introducing a preprocessor flag for selecting such format.

Omitting the declaration of the pre-processor flag the EVID_TRINAT is taken by default see Midas build options and operation considerations.

Special macros are avaialbe to retrieve this information based on the EVID content and the type of EVID structure.

The Macro parameter should point to the first data of the EVID bank.

```
// check if EVID is present if so display its content
if ((status = ybk_find (pybos, "EVID", &bklen, &bktyp, (void *)&pybk)) == YB_SUCCESS)
```

```
{
  pdata = (DWORD *)((YBOS_BANK_HEADER *)pybk + 1);
  pevent->event_id      = YBOS_EVID_EVENT_ID(pdata);
  pevent->trigger_mask  = YBOS_EVID_TRIGGER_MASK(pdata);
  pevent->serial_number = YBOS_EVID_SERIAL(pdata);
  pevent->time_stamp    = YBOS_EVID_TIME(pdata);
  pevent->data_size     = pybk->length;
}
```

The current type of EVID bank are:

- [EVID_TRINAT] Specific for Trinat experiment.

```
ybk_create((DWORD *)pevent, "EVID", I4_BKTYPE, (DWORD *)(&pbkdat));
*((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
```

- [EVID_TWIST] Specific to Twist Experiment (Triumf).

```
ybk_create((DWORD *)pevent, "EVID", I4_BKTYPE, &pbkdat);
*((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
*(pbkdat)++ = *((DWORD *)frontend_name);     // frontend name
ybk_close((DWORD *)pevent, pbkdat);
```

Definition at line 211 of file ybos.h.

2.25.1.2 #define MIDAS_EVID_BANK(__a, __b, __c, __d, __e)

Value:

```
{\
      DWORD * _pbuf;\
      bk_create(__a, "EVID", TID_DWORD, &_pbuf);\
      *(_pbuf)++ = (DWORD)__b;\
      *(_pbuf)++ = (DWORD)__c;\
      *(_pbuf)++ = (DWORD)__d;\
      *(_pbuf)++ = (DWORD)ss_millitime();\
      *(_pbuf)++ = (DWORD)__e;\
      bk_close(__a, _pbuf);\
        }
```

pevt Evt# id/msk serial run#

Definition at line 255 of file ybos.h.

2.25.1.3 #define SWAP_D2WORD(_d2w)

Value:

```
{\
  WORD _tmp2;                                   \
  _tmp2                 = *((WORD *)(_d2w));     \
  *((WORD *)(_d2w))     = *(((WORD *)(_d2w))+1); \
  *(((WORD *)(_d2w))+1) = _tmp2;                 \
}
```

word swap (I4=I21I22 -> I4=I22I21

Definition at line 144 of file ybos.h.

2.25.1.4 #define YBOS_EVID_BANK(__a, __b, __c, __d, __e)

Value:

```
{\
      DWORD * _pbuf;\
      ybk_create(__a, "EVID", I4_BKTYPE, &_pbuf);\
      *(_pbuf)++ = (DWORD)__b;\
      *(_pbuf)++ = (DWORD)__c;\
      *(_pbuf)++ = (DWORD)__d;\
      *(_pbuf)++ = (DWORD)ss_millitime();\
      *(_pbuf)++ = (DWORD)__e;\
      ybk_close(__a, _pbuf);\
        }
```

pevt Evt# id/msk serial run#

Definition at line 241 of file ybos.h.

2.25.1.5 #define YBOS_EVID_EVENT_ID(e) ∗((WORD ∗)(e)+1)

Definition at line 215 of file ybos.h.

2.25.1.6 #define YBOS_EVID_EVENT_NB(e) ∗((DWORD ∗)(e)+1)

Definition at line 220 of file ybos.h.

2.25.1.7 #define YBOS_EVID_RUN_NUMBER(e) ∗((DWORD ∗)(e)+3)

Definition at line 219 of file ybos.h.

2.25.1.8 #define YBOS_EVID_SERIAL(e) ∗((DWORD ∗)(e)+1)

Definition at line 217 of file ybos.h.

2.25.1.9 #define YBOS_EVID_TIME(e) $*$((DWORD $*$)(e)+2)

Definition at line 218 of file ybos.h.

2.25.1.10 #define YBOS_EVID_TRIGGER_MASK(e) $*$((WORD $*$)(e)+0)

Definition at line 216 of file ybos.h.

2.26 YBOS error code

Defines

- #define YB_SUCCESS 1
- #define YB_EVENT_NOT_SWAPPED 2
- #define YB_DONE 2
- #define YB_WRONG_BANK_TYPE -100
- #define YB_BANK_NOT_FOUND -101
- #define YB_SWAP_ERROR -102
- #define YB_NOMORE_SLOT -103
- #define YB_UNKNOWN_FORMAT -104

2.26.1 Define Documentation

2.26.1.1 #define YB_BANK_NOT_FOUND -101

Bank not found

Definition at line 85 of file ybos.h.

2.26.1.2 #define YB_DONE 2

Operation complete

Definition at line 83 of file ybos.h.

2.26.1.3 #define YB_EVENT_NOT_SWAPPED 2

Not swapped

Definition at line 82 of file ybos.h.

2.26.1.4 #define YB_NOMORE_SLOT -103

No more space for fragment

Definition at line 87 of file ybos.h.

2.26.1.5 #define YB_SUCCESS 1

Ok

Definition at line 81 of file ybos.h.

2.26.1.6 #define YB_SWAP_ERROR -102

Error swapping

Definition at line 86 of file ybos.h.

2.26.1.7 #define YB_UNKNOWN_FORMAT -104

Unknown format (see YBOS format)

Definition at line 88 of file ybos.h.

2.26.1.8 #define YB_WRONG_BANK_TYPE -100

Wrong bank type (see YBOS Bank Types)

Definition at line 84 of file ybos.h.

2.27 YBOS Bank Functions (ybk_xxx)

Functions

- void ybk_init (DWORD ∗plrl)
- void ybk_create (DWORD ∗plrl, char ∗bkname, DWORD bktype, void ∗pbkdat)
- INT ybk_close (DWORD ∗plrl, void ∗pbkdat)
- INT ybk_size (DWORD ∗plrl)
- INT ybk_list (DWORD ∗plrl, char ∗bklist)
- INT ybk_find (DWORD ∗plrl, char ∗bkname, DWORD ∗bklen, DWORD ∗bktype, void ∗∗pbk)
- INT ybk_locate (DWORD ∗plrl, char ∗bkname, void ∗pdata)
- INT ybk_iterate (DWORD ∗plrl, YBOS_BANK_HEADER ∗∗pybkh, void ∗∗pdata)

2.27.1 Function Documentation

2.27.1.1 INT ybk_close (DWORD ∗ *plrl*, void ∗ *pbkdat*)

Close the YBOS bank previously created by ybk_create().

The data pointer pdata must be obtained by ybk_create() and used as an address to fill a bank. It is incremented with every value written to the bank and finally points to a location just after the last byte of the bank. It is then passed to ybk_close() to finish the bank creation. YBOS is a 4 bytes bank aligned structure. Padding is performed at the closing of the bank with values of 0x0f or/and 0x0ffb. See YBOS bank examples.

Parameters:

plrl pointer to current composed event.

pbkdat pointer to the current data.

Returns:

number number of bytes contained in bank.

Definition at line 386 of file ybos.c.

2.27.1.2 void ybk_create (DWORD ∗ *plrl*, char ∗ *bkname*, DWORD *bktype*, void ∗ *pbkdat*)

Define the following memory area to be a YBOS bank with the given attribute. See YBOS bank examples.

Before banks can be created in an event, ybk_init(). has to be called first. YBOS does not support mixed bank type. i.e: all the data are expected to be of the same type. YBOS is a 4 bytes bank aligned structure. Padding is performed at the closing of the bank (see ybk_close) with values of 0x0f or/and 0x0ffb. See YBOS bank examples.

Parameters:

plrl pointer to the first DWORD of the event area.

bkname name to be assigned to the breated bank (max 4 char)

bktype YBOS Bank Types of the values for the entire created bank.

pbkdat return pointer to the first empty data location.

Returns:

void

Definition at line 273 of file ybos.c.

2.27.1.3 INT ybk_find (DWORD $*$ *plrl*, char $*$ *bkname*, DWORD $*$ *bklen*, DWORD $*$ *bktype*, void $**$ *pbk*)

Find the requested bank and return the infirmation if the bank as well as the pointer to the top of the data section.

Parameters:

plrl pointer to the area of event.

bkname name of the bank to be located.

bklen returned length in 4bytes unit of the bank.

bktype returned bank type.

pbk pointer to the first data of the found bank.

Returns:

YB_SUCCESS, YB_BANK_NOT_FOUND, YB_WRONG_BANK_TYPE

Definition at line 480 of file ybos.c.

2.27.1.4 void ybk_init (DWORD $*$ *plrl*)

Initializes an event for YBOS banks structure.

Before banks can be created in an event, ybk_init() has to be called first. See YBOS bank examples.

Parameters:

plrl pointer to the first DWORD of the event area of event

Returns:

void

Definition at line 243 of file ybos.c.

2.27.1.5 INT ybk_iterate (DWORD $*$ *plrl*, YBOS_BANK_HEADER $**$ *pybkh*, void $**$ *pdata*)

Returns the bank header pointer and data pointer of the given bank name.

Parameters:

plrl pointer to the area of event.

pybkh pointer to the YBOS bank header.

pdata pointer to the first data of the current bank.

Returns:

data length in 4 bytes unit. return -1 if no more bank found.

Definition at line 566 of file ybos.c.

Referenced by update_odb().

2.27.1.6 INT ybk_list (DWORD ∗ *plrl*, char ∗ *bklist*)

Returns the size in bytes of the event composed of YBOS bank(s).

The bk_list() has to be a predefined string of max size of YB_STRING_BANKLIST_MAX.

Parameters:

plrl pointer to the area of event

bklist Filled character string of the YBOS bank names found in the event.

Returns:

number of banks found in this event.

Definition at line 431 of file ybos.c.

2.27.1.7 INT ybk_locate (DWORD ∗ *plrl*, char ∗ *bkname*, void ∗ *pdata*)

Locate the requested bank and return the pointer to the top of the data section.

Parameters:

plrl pointer to the area of event

bkname name of the bank to be located.

pdata pointer to the first data of the located bank.

Returns:

Number of DWORD in bank or YB_BANK_NOT_FOUND, YB_WRONG_BANK_TYPE (<0)

Definition at line 527 of file ybos.c.

2.27.1.8 INT ybk_size (DWORD ∗ *plrl*)

Returns the size in bytes of the event composed of YBOS bank(s).

Parameters:

plrl pointer to the area of event

Returns:

number of bytes contained in data area of the event

Definition at line 416 of file ybos.c.

2.28 Midas Common Functions (cm_xxx)

Data Structures

- struct TR_CLIENT

Functions

- INT cm_synchronize (DWORD ∗seconds)
- INT cm_asctime (char ∗str, INT buf_size)
- INT cm_time (DWORD ∗t)
- char ∗ cm_get_version ()
- int cm_get_revision ()
- INT cm_set_path (char ∗path)
- INT cm_get_path (char ∗path)
- INT cm_scan_experiments (void)
- INT cm_delete_client_info (HNDLE hDB, INT pid)
- INT cm_check_client (HNDLE hDB, HNDLE hKeyClient)
- INT cm_set_client_info (HNDLE hDB, HNDLE ∗hKeyClient, char ∗host_name, char ∗client_name, INT hw_type, char ∗password, DWORD watchdog_timeout)
- INT cm_get_client_info (char ∗client_name)
- INT cm_get_environment (char ∗host_name, int host_name_size, char ∗exp_name, int exp_name_size)
- INT cm_connect_experiment (char ∗host_name, char ∗exp_name, char ∗client_name, void(∗func)(char ∗))
- INT cm_connect_experiment1 (char ∗host_name, char ∗exp_name, char ∗client_name, void(∗func)(char ∗), INT odb_size, DWORD watchdog_timeout)
- INT cm_list_experiments (char ∗host_name, char exp_name[MAX_EXPERIMENT][NAME_LENGTH])
- INT cm_select_experiment (char ∗host_name, char ∗exp_name)
- INT cm_connect_client (char ∗client_name, HNDLE ∗hConn)
- INT cm_disconnect_client (HNDLE hConn, BOOL bShutdown)
- INT cm_disconnect_experiment (void)
- INT cm_set_experiment_database (HNDLE hDB, HNDLE hKeyClient)
- INT cm_get_experiment_database (HNDLE ∗hDB, HNDLE ∗hKeyClient)
- int bm_validate_client_index (const BUFFER ∗buf)
- INT cm_set_watchdog_params (BOOL call_watchdog, DWORD timeout)
- INT cm_get_watchdog_params (BOOL ∗call_watchdog, DWORD ∗timeout)
- INT cm_get_watchdog_info (HNDLE hDB, char ∗client_name, DWORD ∗timeout, DWORD ∗last)
- INT cm_register_transition (INT transition, INT(∗func)(INT, char ∗), INT sequence_number)

- INT cm_set_transition_sequence (INT transition, INT sequence_number)
- INT cm_register_deferred_transition (INT transition, BOOL(∗func)(INT, BOOL))
- INT cm_check_deferred_transition ()
- INT cm_transition (INT transition, INT run_number, char ∗errstr, INT errstr_size, INT async_flag, INT debug_flag)
- INT cm_yield (INT millisec)
- INT cm_execute (char ∗command, char ∗result, INT bufsize)
- INT cm_shutdown (char ∗name, BOOL bUnique)
- INT cm_exist (char ∗name, BOOL bUnique)
- INT cm_cleanup (char ∗client_name, BOOL ignore_timeout)

2.28.1 Function Documentation

2.28.1.1 int bm_validate_client_index (const BUFFER ∗ *buf*) `[static]`

dox∗∗∗

Definition at line 2304 of file midas.c.

Referenced by bm_close_buffer(), bm_empty_buffers(), bm_flush_cache(), bm_push_event(), bm_receive_event(), bm_remove_event_request(), bm_send_event(), bm_skip_event(), bm_wait_for_free_space(), cm_cleanup(), and cm_set_watchdog_params().

2.28.1.2 INT cm_asctime (char ∗ *str*, INT *buf_size*)

Get time from MIDAS server and set local time.

Parameters:

str return time string

buf_size Maximum size of str

Returns:

CM_SUCCESS

Definition at line 979 of file midas.c.

Referenced by al_trigger_alarm(), and cm_transition().

2.28.1.3 INT cm_check_client (HNDLE *hDB*, HNDLE *hKeyClient*)

Check if a client with a /system/client/xxx entry has a valid entry in the ODB client table. If not, remove that client from the /system/client tree.

Parameters:

hDB Handle to online database

hKeyClient Handle to client key

Returns:

CM_SUCCESS, CM_NO_CLIENT

Definition at line 1257 of file midas.c.

Referenced by cm_set_client_info().

2.28.1.4 INT cm_check_deferred_transition ()

Check for any deferred transition. If a deferred transition handler has been registered via the cm_register_deferred_transition function, this routine should be called regularly. It checks if a transition request is pending. If so, it calld the registered handler if the transition should be done and then actually does the transition.

Returns:

CM_SUCCESS, <error> Error from cm_transition()

Definition at line 2918 of file midas.c.

Referenced by scheduler().

2.28.1.5 INT cm_cleanup (char $*$ *client_name*, BOOL *ignore_timeout*)

Remove hanging clients independent of their watchdog timeout.

Since this function does not obey the client watchdog timeout, it should be only called to remove clients which have their watchdog checking turned off or which are known to be dead. The normal client removement is done via cm_watchdog().

Currently (Sept. 02) there are two applications for that:

1. The ODBEdit command "cleanup", which can be used to remove clients which have their watchdog checking off, like the analyzer started with the "-d" flag for a debugging session.
2. The frontend init code to remove previous frontends. This can be helpful if a frontend dies. Normally, one would have to wait 60 sec. for a crashed frontend to be removed. Only then one can start again the frontend. Since the frontend init code contains a call to cm_cleanup(<frontend_name>), one can restart a frontend immediately.

Added ignore_timeout on Nov.03. A logger might have an increased tiemout of up to 60 sec. because of tape operations. If ignore_timeout is FALSE, the logger is then not killed if its inactivity is less than 60 sec., while in the previous implementation it was always killed after 2∗WATCHDOG_INTERVAL.

Parameters:

client_name Client name, if zero check all clients

ignore_timeout If TRUE, ignore a possible increased timeout defined by each client.

Returns:

CM_SUCCESS

Definition at line 4561 of file midas.c.

Referenced by main().

2.28.1.6 INT cm_connect_client (char ∗ *client_name*, HNDLE ∗ *hConn*)

Connect to a MIDAS client of the current experiment

For Internal use only.

Parameters:

client_name Name of client to connect to. This name is set by the other client via the cm_connect_experiment call.

hConn Connection handle

Returns:

CM_SUCCESS, CM_NO_CLIENT

Definition at line 2024 of file midas.c.

2.28.1.7 INT cm_connect_experiment (char ∗ *host_name*, char ∗ *exp_name*, char ∗ *client_name*, void(∗)(char ∗) *func*)

This function connects to an existing MIDAS experiment. This must be the first call in a MIDAS application. It opens three TCP connection to the remote host (one for RPC calls, one to send events and one for hot-link notifications from the remote host) and writes client information into the ODB under /System/Clients.

Attention:

All MIDAS applications should evaluate the MIDAS_SERVER_HOST and MIDAS_EXPT_NAME environment variables as defaults to the host name and experiment name (see Environment variables). For that purpose, the function cm_get_environment() should be called prior to cm_connect_experiment(). If command line parameters -h and -e are used, the evaluation should be done between cm_get_environment() and cm_connect_experiment(). The function cm_disconnect_experiment() must be called before a MIDAS application exits.

```
#include <stdio.h>
#include <midas.h>
main(int argc, char *argv[])
{
  INT  status, i;
  char host_name[256],exp_name[32];

  // get default values from environment
  cm_get_environment(host_name, exp_name);

  // parse command line parameters
  for (i=1 ; i<argc ; i++)
    {
    if (argv[i][0] == '-')
      {
      if (i+1 >= argc || argv[i+1][0] == '-')
        goto usage;
      if (argv[i][1] == 'e')
        strcpy(exp_name, argv[++i]);
      else if (argv[i][1] == 'h')
        strcpy(host_name, argv[++i]);
      else
        {
usage:
        printf("usage: test [-h Hostname] [-e Experiment]\n\n");
        return 1;
        }
      }
    }
  status = cm_connect_experiment(host_name, exp_name, "Test", NULL);
  if (status != CM_SUCCESS)
    return 1;
  ...do operations...
  cm_disconnect_experiment();
}
```

Parameters:

host_name Specifies host to connect to. Must be a valid IP host name. The string can be empty ("") if to connect to the local computer.

exp_name Specifies the experiment to connect to. If this string is empty, the number of defined experiments in exptab is checked. If only one experiment is defined, the function automatically connects to this one. If more than one experiment is defined, a list is presented and the user can interactively select one experiment.

client_name Client name of the calling program as it can be seen by others (like the scl command in ODBEdit).

func Callback function to read in a password if security has been enabled. In all command line applications this function is NULL which invokes an internal ss_gets() function to read in a password. In windows environments (MS Windows, X Windows) a function can be supplied to open a dialog box and read in the password. The argument of this function must be the returned password.

Returns:

CM_SUCCESS, CM_UNDEF_EXP, CM_SET_ERROR, RPC_NET_ERROR CM_VERSION_MISMATCH MIDAS library version different on local and remote computer

Definition at line 1664 of file midas.c.

Referenced by main().

2.28.1.8 INT cm_connect_experiment1 (char ∗ *host_name*, char ∗ *exp_name*, char ∗ *client_name*, void(∗)(char ∗) *func*, INT *odb_size*, DWORD *watchdog_timeout*)

Connect to a MIDAS experiment (to the online database) on a specific host.

For internal use only.

Definition at line 1685 of file midas.c.

Referenced by cm_connect_experiment(), and main().

2.28.1.9 INT cm_delete_client_info (HNDLE *hDB*, INT *pid*)

Delete client info from database

Parameters:

hDB Database handle

pid PID of entry to delete, zero for this process.

Returns:

CM_SUCCESS

Definition at line 1206 of file midas.c.

Referenced by cm_check_client(), cm_cleanup(), and cm_disconnect_experiment().

2.28.1.10 INT cm_deregister_transition (INT *transition*)

Definition at line 2738 of file midas.c.

2.28.1.11 INT cm_disconnect_client (HNDLE *hConn*, BOOL *bShutdown*)

Disconnect from a MIDAS client

Parameters:

hConn Connection handle obtained via cm_connect_client()

bShutdown If TRUE, disconnect from client and shut it down (exit the client program) by sending a RPC_SHUTDOWN message

Returns:

see rpc_client_disconnect()

Definition at line 2090 of file midas.c.

2.28.1.12 INT cm_disconnect_experiment (void)

Disconnect from a MIDAS experiment.

Attention:

Should be the last call to a MIDAS library function in an application before it exits. This function removes the client information from the ODB, disconnects all TCP connections and frees all internal allocated memory. See cm_connect_experiment() for example.

Returns:

CM_SUCCESS

Definition at line 2104 of file midas.c.

Referenced by cm_connect_experiment1(), main(), and register_equipment().

2.28.1.13 INT cm_execute (char ∗ *command*, char ∗ *result*, INT *bufsize*)

Executes command via system() call

Parameters:

command Command string to execute

result stdout of command

bufsize string size in byte

Returns:

CM_SUCCESS

Definition at line 3634 of file midas.c.

2.28.1.14 INT cm_exist (char ∗ *name*, BOOL *bUnique*)

Check if a MIDAS client exists in current experiment

Parameters:

name Client name

bUnique If true, look for the exact client name. If false, look for namexxx where xxx is a any number

Returns:

CM_SUCCESS, CM_NO_CLIENT

Definition at line 4483 of file midas.c.

Referenced by al_check(), and main().

2.28.1.15 INT cm_get_client_info (char ∗ *client_name*)

Get info about the current client

Parameters:

∗***client_name*** Client name.

Returns:

CM_SUCCESS, CM_UNDEF_EXP

Definition at line 1491 of file midas.c.

Referenced by bm_open_buffer().

2.28.1.16 INT cm_get_environment (char ∗ *host_name*, int *host_name_size*, char ∗ *exp_name*, int *exp_name_size*)

Returns MIDAS environment variables.

Attention:

This function can be used to evaluate the standard MIDAS environment variables before connecting to an experiment (see Environment variables). The usual way is that the host name and experiment name are first derived from the environment variables MIDAS_SERVER_HOST and MIDAS_EXPT_NAME. They can then be superseded by command line parameters with -h and -e flags.

```
#include <stdio.h>
#include <midas.h>
main(int argc, char *argv[])
{
  INT  status, i;
  char host_name[256],exp_name[32];

  // get default values from environment
  cm_get_environment(host_name, exp_name);

  // parse command line parameters
  for (i=1 ; i<argc ; i++)
    {
```

```
    if (argv[i][0] == '-')
      {
      if (i+1 >= argc || argv[i+1][0] == '-')
        goto usage;
      if (argv[i][1] == 'e')
        strcpy(exp_name, argv[++i]);
      else if (argv[i][1] == 'h')
        strcpy(host_name, argv[++i]);
      else
        {
usage:
        printf("usage: test [-h Hostname] [-e Experiment]\n\n");
        return 1;
        }
      }
    }
  status = cm_connect_experiment(host_name, exp_name, "Test", NULL);
  if (status != CM_SUCCESS)
    return 1;
    ...do anyting...
  cm_disconnect_experiment();
}
```

Parameters:

host_name Contents of MIDAS_SERVER_HOST environment variable.

host_name_size string length

exp_name Contents of MIDAS_EXPT_NAME environment variable.

exp_name_size string length

Returns:

CM_SUCCESS

Definition at line 1567 of file midas.c.

Referenced by main().

2.28.1.17 INT cm_get_experiment_database (HNDLE ∗ *hDB*, HNDLE ∗ *hKeyClient*)

Get the handle to the ODB from the currently connected experiment.

Attention:

This function returns the handle of the online database (ODB) which can be used in future db_xxx() calls. The hkeyclient key handle can be used to access the client information in the ODB. If the client key handle is not needed, the parameter can be NULL.

```
HNDLE hDB, hkeyclient;
 char  name[32];
 int   size;
```

```
db_get_experiment_database(&hdb, &hkeyclient);
size = sizeof(name);
db_get_value(hdb, hkeyclient, "Name", name, &size, TID_STRING, TRUE);
printf("My name is %s\n", name);
```

Parameters:

hDB Database handle.

hKeyClient Handle for key where search starts, zero for root.

Returns:

CM_SUCCESS

Definition at line 2250 of file midas.c.

Referenced by al_check(), al_reset_alarm(), al_trigger_alarm(), ana_end_of_run(), analyzer_init(), bm_open_buffer(), cm_connect_client(), cm_deregister_transition(), cm_disconnect_experiment(), cm_exist(), cm_get_client_info(), cm_msg_log(), cm_msg_log1(), cm_msg_retrieve(), cm_register_deferred_transition(), cm_register_transition(), cm_set_transition_sequence(), cm_set_watchdog_params(), cm_shutdown(), cm_transition(), el_submit(), and main().

2.28.1.18 INT cm_get_path (char ∗ *path*)

Return the path name previously set with cm_set_path.

Parameters:

path Pathname

Returns:

CM_SUCCESS

Definition at line 1077 of file midas.c.

Referenced by cm_connect_experiment1(), cm_msg_log(), cm_msg_log1(), and cm_msg_retrieve().

2.28.1.19 int cm_get_revision ()

Return svn revision number of current MIDAS library as a string

Returns:

revision number

Definition at line 1046 of file midas.c.

2.28.1.20 char∗ cm_get_version ()

Return version number of current MIDAS library as a string

Returns:
version number

Definition at line 1037 of file midas.c.

2.28.1.21 INT cm_get_watchdog_info (HNDLE *hDB*, char ∗ *client_name*, DWORD ∗ *timeout*, DWORD ∗ *last*)

Return watchdog information about specific client

Parameters:

hDB ODB handle

client_name ODB client name

timeout Timeout for this application in seconds

last Last time watchdog was called in msec

Returns:
CM_SUCCESS, CM_NO_CLIENT, DB_INVALID_HANDLE

Definition at line 2515 of file midas.c.

2.28.1.22 INT cm_get_watchdog_params (BOOL ∗ *call_watchdog*, DWORD ∗ *timeout*)

Return the current watchdog parameters

Parameters:

call_watchdog Call the cm_watchdog routine periodically

timeout Timeout for this application in seconds

Returns:
CM_SUCCESS

Definition at line 2495 of file midas.c.

Referenced by bm_open_buffer(), cm_connect_experiment1(), cm_set_client_info(), and db_open_database().

2.28.1.23 INT cm_list_experiments (char ∗ *host_name*, char *exp_name*[MAX_EXPERIMENT][NAME_LENGTH])

Connect to a MIDAS server and return all defined experiments in ∗exp_name[MAX_EXPERIMENTS]

Parameters:

host_name Internet host name.

exp_name list of experiment names

Returns:

CM_SUCCESS, RPC_NET_ERROR

Definition at line 1882 of file midas.c.

Referenced by cm_select_experiment().

2.28.1.24 INT cm_register_deferred_transition (INT *transition*, BOOL(∗)(INT, BOOL) *func*)

Register a deferred transition handler. If a client is registered as a deferred transition handler, it may defer a requested transition by returning FALSE until a certain condition (like a motor reaches its end position) is reached.

Parameters:

transition One of TR_xxx

(∗func) Function which gets called whenever a transition is requested. If it returns FALSE, the transition is not performed.

Returns:

CM_SUCCESS, <error> Error from ODB access

Definition at line 2861 of file midas.c.

2.28.1.25 INT cm_register_transition (INT *transition*, INT(∗)(INT, char ∗) *func*, INT *sequence_number*)

Registers a callback function for run transitions. This function internally registers the transition callback function and publishes its request for transition notification by writing a transition request to /System/Clients/<pid>/Transition XXX. Other clients making a transition scan the transition requests of all clients and call their transition callbacks via RPC.

Clients can register for transitions (Start/Stop/Pause/Resume) in a given sequence. All sequence numbers given in the registration are sorted on a transition and the clients are contacted in ascending order. By default, all programs register with a sequence number of 500. The logger however uses 200 for start, so that it can open files before the other clients are contacted, and 800 for stop, so that the files get closed when all other clients have gone already through the stop trantition.

The callback function returns CM_SUCCESS if it can perform the transition or a value larger than one in case of error. An error string can be copied into the error variable.

Attention:

The callback function will be called on transitions from inside the cm_yield() function which therefore must be contained in the main program loop.

```
INT start(INT run_number, char *error)
{
  if (<not ok>)
    {
    strcpy(error, "Cannot start because ...");
    return 2;
    }
  printf("Starting run %d\n", run_number);
  return CM_SUCCESS;
}
main()
{
  ...
  cm_register_transition(TR_START, start, 500);
  do
    {
    status = cm_yield(1000);
    } while (status != RPC_SHUTDOWN &&
             status != SS_ABORT);
  ...
}
```

Parameters:

transition Transition to register for (see State Codes & Transition Codes)

func Callback function.

sequence_number Sequence number for that transition (1..1000)

Returns:

CM_SUCCESS

Definition at line 2674 of file midas.c.

Referenced by main().

2.28.1.26 INT cm_scan_experiments (void)

Scan the "exptab" file for MIDAS experiment names and save them for later use by rpc_server_accept(). The file is first searched under $MIDAS/exptab if present, then the directory from argv[0] is probed.

Returns:

CM_SUCCESS
CM_UNDEF_EXP exptab not found and MIDAS_DIR not set

Definition at line 1113 of file midas.c.

Referenced by cm_connect_experiment1(), and cm_list_experiments().

2.28.1.27 INT cm_select_experiment (char ∗ *host_name*, char ∗ *exp_name*)

Connect to a MIDAS server and select an experiment from the experiments available on this server

For Internal use only.

Parameters:

host_name Internet host name.

exp_name list of experiment names

Returns:

CM_SUCCESS, RPC_NET_ERROR

Definition at line 1985 of file midas.c.

Referenced by cm_connect_experiment1().

2.28.1.28 INT cm_set_client_info (HNDLE *hDB*, HNDLE ∗ *hKeyClient*, char ∗ *host_name*, char ∗ *client_name*, INT *hw_type*, char ∗ *password*, DWORD *watchdog_timeout*)

Set client information in online database and return handle

Parameters:

hDB Handle to online database

hKeyClient returned key

host_name server name

client_name Name of this program as it will be seen by other clients.

hw_type Type of byte order

password MIDAS password

watchdog_timeout Default watchdog timeout, can be overwritten by ODB setting /programs/<name>/Watchdog timeout

Returns:

CM_SUCCESS

Definition at line 1319 of file midas.c.

Referenced by cm_connect_experiment1().

2.28.1.29 INT cm_set_experiment_database (HNDLE *hDB*, HNDLE *hKeyClient*)

Set the handle to the ODB for the currently connected experiment

Parameters:

hDB Database handle

hKeyClient Key handle of client structure

Returns:

CM_SUCCESS

Definition at line 2185 of file midas.c.

Referenced by cm_connect_experiment1(), and cm_disconnect_experiment().

2.28.1.30 INT cm_set_path (char ∗ *path*)

Set path to actual experiment. This function gets called by cm_connect_experiment if the connection is established to a local experiment (not through the TCP/IP server). The path is then used for all shared memory routines.

Parameters:

path Pathname

Returns:

CM_SUCCESS

Definition at line 1060 of file midas.c.

Referenced by cm_connect_experiment1().

2.28.1.31 INT cm_set_transition_sequence (INT *transition*, INT *sequence_number*)

Change the transition sequence for the calling program.

Parameters:

transition TR_START, TR_PAUSE, TR_RESUME or TR_STOP.

sequence_number New sequence number, should be between 1 and 1000

Returns:

CM_SUCCESS

Definition at line 2797 of file midas.c.

2.28.1.32 INT cm_set_watchdog_params (BOOL *call_watchdog*, DWORD *timeout*)

Sets the internal watchdog flags and the own timeout. If call_watchdog is TRUE, the cm_watchdog routine is called periodically from the system to show other clients that this application is "alive". On UNIX systems, the alarm() timer is used which is then not available for user purposes.

The timeout specifies the time, after which the calling application should be considered "dead" by other clients. Normally, the cm_watchdog() routines is called periodically. If a client crashes, this does not occur any more. Then other clients can detect this and clear all buffer and database entries of this application so they are not blocked any more. If this application should not checked by others, the timeout can be specified as zero. It might be useful for debugging purposes to do so, because if a debugger comes to a breakpoint and stops the application, the periodic call of cm_watchdog is disabled and the client looks like dead.

If the timeout is not zero, but the watchdog is not called (call_watchdog == FALSE), the user must ensure to call cm_watchdog periodically with a period of WATCHDOG_-INTERVAL milliseconds or less.

An application which calles system routines which block the alarm signal for some time, might increase the timeout to the maximum expected blocking time before issuing the calls. One example is the logger doing Exabyte tape IO, which can take up to one minute.

Parameters:

call_watchdog Call the cm_watchdog routine periodically

timeout Timeout for this application in ms

Returns:

CM_SUCCESS

Definition at line 2383 of file midas.c.

Referenced by cm_connect_experiment1(), cm_set_client_info(), and main().

2.28.1.33 INT cm_shutdown (char ∗ *name*, BOOL *bUnique*)

Shutdown (exit) other MIDAS client

Parameters:

name Client name or "all" for all clients

bUnique If true, look for the exact client name. If false, look for namexxx where xxx is a any number.

Returns:

CM_SUCCESS, CM_NO_CLIENT, DB_NO_KEY

Definition at line 4394 of file midas.c.

Referenced by cm_transition(), and main().

2.28.1.34 INT cm_synchronize (DWORD ∗ *seconds*)

Get time from MIDAS server and set local time.

Parameters:

seconds Time in seconds

Returns:

CM_SUCCESS

Definition at line 951 of file midas.c.

Referenced by main().

2.28.1.35 INT cm_time (DWORD ∗ *t*)

Get time from ss_time on server.

Parameters:

t string

Returns:

CM_SUCCESS

Definition at line 997 of file midas.c.

Referenced by cm_transition().

2.28.1.36 INT cm_transition (INT *transition*, INT *run_number*, char ∗ *errstr*, INT *errstr_size*, INT *async_flag*, INT *debug_flag*)

Performs a run transition (Start/Stop/Pause/Resume).

Synchronous/Asynchronous flag. If set to ASYNC, the transition is done asynchronously, meaning that clients are connected and told to execute their callback routine, but no result is awaited. The return value is specified by the transition callback function on the remote clients. If all callbacks can perform the transition, CM_SUCCESS is returned. If one callback cannot perform the transition, the return value of this callback is returned from cm_transition(). The async_flag is usually FALSE so that transition callbacks can block a run transition in case of problems and return an error string. The only exception are situations where a run transition is performed automatically by a program which cannot block in a transition. For example the logger can cause a run stop when a disk is nearly full but it cannot block in the cm_transition() function since it has its own run stop callback which must flush buffers and close disk files and tapes.

```
...
    i = 1;
    db_set_value(hDB, 0, "/Runinfo/Transition in progress", &i, sizeof(INT), 1, TID_INT);

      status = cm_transition(TR_START, new_run_number, str, sizeof(str), SYNC, debug_flag);
      if (status != CM_SUCCESS)
      {
        // in case of error
        printf("Error: %s\n", str);
      }
    ...
```

Parameters:

transition TR_START, TR_PAUSE, TR_RESUME or TR_STOP.

run_number New run number. If zero, use current run number plus one.

errstr returned error string.

errstr_size Size of error string.

async_flag SYNC: synchronization flag (SYNC:wait completion, ASYNC: retun immediately)

debug_flag If 1 output debugging information, if 2 output via cm_msg().

Returns:

CM_SUCCESS, <error> error code from remote client

Definition at line 3011 of file midas.c.

Referenced by cm_check_deferred_transition(), scan_fragment(), and scheduler().

2.28.1.37 INT cm_yield (INT *millisec*)

Central yield functions for clients. This routine should be called in an infinite loop by a client in order to give the MIDAS system the opportunity to receive commands over RPC channels, update database records and receive events.

Parameters:

millisec Timeout in millisec. If no message is received during the specified timeout, the routine returns. If millisec=-1, it only returns when receiving an RPC_SHUTDOWN message.

Returns:

CM_SUCCESS, RPC_SHUTDOWN

Definition at line 3581 of file midas.c.

Referenced by scan_fragment(), and scheduler().

2.28.1.38 int tr_compare (const void ∗ *arg1*, const void ∗ *arg2*)

Definition at line 2966 of file midas.c.

Referenced by cm_transition().

2.29 Midas Buffer Manager Functions (bm_xxx)

Functions

- INT bm_match_event (short int event_id, short int trigger_mask, EVENT_HEADER ∗pevent)
- INT bm_open_buffer (char ∗buffer_name, INT buffer_size, INT ∗buffer_handle)
- INT bm_close_buffer (INT buffer_handle)
- INT bm_close_all_buffers (void)
- INT bm_set_cache_size (INT buffer_handle, INT read_size, INT write_size)
- INT bm_compose_event (EVENT_HEADER ∗event_header, short int event_id, short int trigger_mask, DWORD size, DWORD serial)
- INT bm_request_event (HNDLE buffer_handle, short int event_id, short int trigger_mask, INT sampling_type, HNDLE ∗request_id, void(∗func)(HNDLE, HNDLE, EVENT_HEADER ∗, void ∗))
- INT bm_remove_event_request (INT buffer_handle, INT request_id)
- INT bm_delete_request (INT request_id)
- INT bm_send_event (INT buffer_handle, void ∗source, INT buf_size, INT async_flag)
- INT bm_flush_cache (INT buffer_handle, INT async_flag)
- INT bm_receive_event (INT buffer_handle, void ∗destination, INT ∗buf_size, INT async_flag)
- INT bm_skip_event (INT buffer_handle)
- INT bm_push_event (char ∗buffer_name)
- INT bm_check_buffers ()
- INT bm_empty_buffers ()

2.29.1 Function Documentation

2.29.1.1 INT bm_check_buffers ()

Check if any requested event is waiting in a buffer

Returns:

TRUE More events are waiting
FALSE No more events are waiting

Definition at line 6803 of file midas.c.

Referenced by cm_yield().

2.29.1.2 INT bm_close_all_buffers (void)

Close all open buffers

Returns:

BM_SUCCESS

Definition at line 4136 of file midas.c.

Referenced by cm_disconnect_experiment(), and cm_set_client_info().

2.29.1.3 INT bm_close_buffer (INT *buffer_handle*)

Closes an event buffer previously opened with bm_open_buffer().

Parameters:

buffer_handle buffer handle

Returns:

BM_SUCCESS, BM_INVALID_HANDLE

Definition at line 4020 of file midas.c.

Referenced by bm_close_all_buffers(), and source_unbooking().

2.29.1.4 INT bm_compose_event (EVENT_HEADER ∗ *event_header*, short int *event_id*, short int *trigger_mask*, DWORD *size*, DWORD *serial*)

Compose a Midas event header. An event header can usually be set-up manually or through this routine. If the data size of the event is not known when the header is composed, it can be set later with event_header->data-size = <...> Following structure is created at the beginning of an event

```
typedef struct {
 short int     event_id;
 short int     trigger_mask;
 DWORD         serial_number;
 DWORD         time_stamp;
 DWORD         data_size;
} EVENT_HEADER;

char event[1000];
 bm_compose_event((EVENT_HEADER *)event, 1, 0, 100, 1);
 *(event+sizeof(EVENT_HEADER)) = <...>
```

Parameters:

event_header pointer to the event header

event_id event ID of the event

trigger_mask trigger mask of the event

size size if the data part of the event in bytes

serial serial number

Returns:

BM_SUCCESS

Definition at line 5102 of file midas.c.

Referenced by cm_msg(), cm_msg1(), and source_scan().

2.29.1.5 void bm_convert_event_header (EVENT_HEADER ∗ *pevent*, int *convert_flags*) `[static]`

Definition at line 5610 of file midas.c.

Referenced by bm_copy_from_cache(), and bm_receive_event().

2.29.1.6 int bm_copy_from_cache (BUFFER ∗ *pbuf*, void ∗ *destination*, int *max_size*, int ∗ *buf_size*, int *convert_flags*) `[static]`

Definition at line 5622 of file midas.c.

Referenced by bm_receive_event().

2.29.1.7 INT bm_delete_request (INT *request_id*)

Deletes an event request previously done with bm_request_event(). When an event request gets deleted, events of that requested type are not received any more. When a buffer is closed via bm_close_buffer(), all event requests from that buffer are deleted automatically

Parameters:

request_id request identifier given by bm_request_event()

Returns:

BM_SUCCESS, BM_INVALID_HANDLE

Definition at line 5396 of file midas.c.

Referenced by bm_close_buffer(), and source_unbooking().

2.29.1.8 void bm_dispatch_event (int *buffer_handle*, EVENT_HEADER ∗ *pevent*) `[static]`

Definition at line 5574 of file midas.c.

Referenced by bm_dispatch_from_cache(), and bm_push_event().

2.29.1.9 void bm_dispatch_from_cache (BUFFER ∗ *pbuf*, int *buffer_handle*) `[static]`

Definition at line 5590 of file midas.c.

Referenced by bm_push_event().

2.29.1.10 INT bm_empty_buffers ()

Clears event buffer and cache. If an event buffer is large and a consumer is slow in analyzing events, events are usually received some time after they are produced. This effect is even more experienced if a read cache is used (via bm_set_cache_size()). When changes to the hardware are made in the experience, the consumer will then still analyze old events before any new event which reflects the hardware change. Users can be fooled by looking at histograms which reflect the hardware change many seconds after they have been made.

To overcome this potential problem, the analyzer can call bm_empty_buffers() just after the hardware change has been made which skips all old events contained in event buffers and read caches. Technically this is done by forwarding the read pointer of the client. No events are really deleted, they are still visible to other clients like the logger.

Note that the front-end also contains write buffers which can delay the delivery of events. The standard front-end framework mfe.c reduces this effect by flushing all buffers once every second.

Returns:

BM_SUCCESS

Definition at line 7118 of file midas.c.

Referenced by handFlush(), source_booking(), and source_unbooking().

2.29.1.11 INT bm_flush_cache (INT *buffer_handle*, INT *async_flag*)

Empty write cache. This function should be used if events in the write cache should be visible to the consumers immediately. It should be called at the end of each run, otherwise events could be kept in the write buffer and will flow to the data of the next run.

Parameters:

buffer_handle Buffer handle obtained via bm_open_buffer()

async_flag Synchronous/asynchronous flag. If FALSE, the function blocks if the buffer has not enough free space to receive the full cache. If TRUE, the function returns immediately with a value of BM_ASYNC_RETURN without writing the cache.

Returns:

BM_SUCCESS, BM_INVALID_HANDLE
BM_ASYNC_RETURN Routine called with async_flag == TRUE and buffer has not enough space to receive cache
BM_NO_MEMORY Event is too large for network buffer or event buffer. One has to increase MAX_EVENT_SIZE in midas.h and recompile.

Definition at line 6065 of file midas.c.

Referenced by bm_send_event(), close_buffers(), scan_fragment(), scheduler(), send_event(), and tr_stop().

2.29.1.12 INT bm_match_event (short int *event_id*, short int *trigger_mask*, EVENT_HEADER ∗ *pevent*)

Check if an event matches a given event request by the event id and trigger mask

Parameters:

event_id Event ID of request

trigger_mask Trigger mask of request

pevent Pointer to event to check

Returns:

TRUE if event matches request

Definition at line 3742 of file midas.c.

Referenced by bm_dispatch_event(), bm_push_event(), bm_receive_event(), bm_send_event(), and bm_wait_for_free_space().

2.29.1.13 INT bm_open_buffer (char ∗ *buffer_name*, INT *buffer_size*, INT ∗ *buffer_handle*)

Open an event buffer. Two default buffers are created by the system. The "SYSTEM" buffer is used to exchange events and the "SYSMSG" buffer is used to exchange system messages. The name and size of the event buffers is defined in midas.h as EVENT_BUFFER_NAME and 2∗MAX_EVENT_SIZE. Following example opens the "SYSTEM" buffer, requests events with ID 1 and enters a main loop. Events are then received in process_event()

```
#include <stdio.h>
#include "midas.h"
void process_event(HNDLE hbuf, HNDLE request_id,
           EVENT_HEADER *pheader, void *pevent)
{
  printf("Received event #%d\r",
  pheader->serial_number);
}
main()
{
  INT status, request_id;
  HNDLE hbuf;
  status = cm_connect_experiment("pc810", "Sample", "Simple Analyzer", NULL);
  if (status != CM_SUCCESS)
  return 1;
  bm_open_buffer(EVENT_BUFFER_NAME, 2*MAX_EVENT_SIZE, &hbuf);
  bm_request_event(hbuf, 1, TRIGGER_ALL, GET_ALL, request_id, process_event);

  do
  {
   status = cm_yield(1000);
  } while (status != RPC_SHUTDOWN && status != SS_ABORT);
  cm_disconnect_experiment();
  return 0;
}
```

Parameters:

buffer_name	Name of buffer
buffer_size	Default size of buffer in bytes. Can by overwritten with ODB value
buffer_handle	Buffer handle returned by function

Returns:

BM_SUCCESS, BM_CREATED
BM_NO_SHM Shared memory cannot be created
BM_NO_MUTEX Mutex cannot be created
BM_NO_MEMORY Not enough memory to create buffer descriptor
BM_MEMSIZE_MISMATCH Buffer size conflicts with an existing buffer of different size
BM_INVALID_PARAM Invalid parameter

Definition at line 3803 of file midas.c.

Referenced by cm_msg(), cm_msg1(), cm_msg_register(), register_equipment(), and source_booking().

2.29.1.14 INT bm_push_event (char ∗ *buffer_name*)

Check a buffer if an event is available and call the dispatch function if found.

Parameters:

buffer_name	Name of buffer

Returns:

BM_SUCCESS, BM_INVALID_HANDLE, BM_TRUNCATED, BM_ASYNC_RETURN, RPC_NET_ERROR

Definition at line 6586 of file midas.c.

Referenced by bm_check_buffers().

2.29.1.15 int bm_read_cache_has_events (const BUFFER ∗ *pbuf*) `[static]`

Definition at line 5656 of file midas.c.

Referenced by bm_push_event(), and bm_receive_event().

2.29.1.16 INT bm_receive_event (INT *buffer_handle*, void ∗ *destination*, INT ∗ *buf_size*, INT *async_flag*)

Receives events directly. This function is an alternative way to receive events without a main loop.

It can be used in analysis systems which actively receive events, rather than using callbacks. A analysis package could for example contain its own command line interface. A command like "receive 1000 events" could make it necessary to call bm_receive_event() 1000 times in a row to receive these events and then return back to the command line prompt. The according bm_request_event() call contains NULL as the callback routine to indicate that bm_receive_event() is called to receive events.

```
#include <stdio.h>
#include "midas.h"
void process_event(EVENT_HEADER *pheader)
{
 printf("Received event #%d\r",
 pheader->serial_number);
}
main()
{
  INT status, request_id;
  HNDLE hbuf;
  char event_buffer[1000];
  status = cm_connect_experiment("", "Sample",
  "Simple Analyzer", NULL);
  if (status != CM_SUCCESS)
   return 1;
  bm_open_buffer(EVENT_BUFFER_NAME, 2*MAX_EVENT_SIZE, &hbuf);
  bm_request_event(hbuf, 1, TRIGGER_ALL, GET_ALL, request_id, NULL);

  do
  {
   size = sizeof(event_buffer);
   status = bm_receive_event(hbuf, event_buffer, &size, ASYNC);
  if (status == CM_SUCCESS)
```

```
   process_event((EVENT_HEADER *) event_buffer);
   <...do something else...>
   status = cm_yield(0);
  } while (status != RPC_SHUTDOWN &&
  status != SS_ABORT);
  cm_disconnect_experiment();
  return 0;
}
```

Parameters:

buffer_handle buffer handle

destination destination address where event is written to

buf_size size of destination buffer on input, size of event plus header on return.

async_flag Synchronous/asynchronous flag. If FALSE, the function blocks if no event is available. If TRUE, the function returns immediately with a value of BM_ASYNC_RETURN without receiving any event.

Returns:

BM_SUCCESS, BM_INVALID_HANDLE
BM_TRUNCATED The event is larger than the destination buffer and was therefore truncated
BM_ASYNC_RETURN No event available

Definition at line 6259 of file midas.c.

Referenced by handFlush(), and source_scan().

2.29.1.17 INT bm_remove_event_request (INT *buffer_handle*, INT *request_id*)

Delete a previously placed request for a specific event type in the client structure of the buffer refereced by buffer_handle.

Parameters:

buffer_handle Handle to the buffer where the re- quest should be placed in

request_id Request id returned by bm_request_event

Returns:

BM_SUCCESS, BM_INVALID_HANDLE, BM_NOT_FOUND, RPC_NET_ERROR

Definition at line 5327 of file midas.c.

Referenced by bm_delete_request().

2.29.1.18 INT bm_request_event (HNDLE *buffer_handle*, short int *event_id*, short int *trigger_mask*, INT *sampling_type*, HNDLE ∗ *request_id*, void(∗)(HNDLE, HNDLE, EVENT_HEADER ∗, void ∗) *func*)

Place an event request based on certain characteristics. Multiple event requests can be placed for each buffer, which are later identified by their request ID. They can contain different callback routines. Example see bm_open_buffer() and bm_receive_event()

Parameters:

buffer_handle buffer handle obtained via bm_open_buffer()

event_id event ID for requested events. Use EVENTID_ALL to receive events with any ID.

trigger_mask trigger mask for requested events. The requested events must have at least one bit in its trigger mask common with the requested trigger mask. Use TRIGGER_ALL to receive events with any trigger mask.

sampling_type specifies how many events to receive. A value of GET_ALL receives all events which match the specified event ID and trigger mask. If the events are consumed slower than produced, the producer is automatically slowed down. A value of GET_SOME receives as much events as possible without slowing down the producer. GET_ALL is typically used by the logger, while GET_SOME is typically used by analyzers.

request_id request ID returned by the function. This ID is passed to the callback routine and must be used in the bm_delete_request() routine.

func allback routine which gets called when an event of the specified type is received.

Returns:

BM_SUCCESS, BM_INVALID_HANDLE
BM_NO_MEMORY too many requests. The value MAX_EVENT_REQUESTS in midas.h should be increased.

Definition at line 5263 of file midas.c.

Referenced by cm_msg_register(), and source_booking().

2.29.1.19 INT bm_send_event (INT *buffer_handle*, void ∗ *source*, INT *buf_size*, INT *async_flag*)

Sends an event to a buffer. This function check if the buffer has enough space for the event, then copies the event to the buffer in shared memory. If clients have requests for the event, they are notified via an UDP packet.

```
char event[1000];
// create event with ID 1, trigger mask 0, size 100 bytes and serial number 1
bm_compose_event((EVENT_HEADER *) event, 1, 0, 100, 1);
```

```
// set first byte of event
*(event+sizeof(EVENT_HEADER)) = <...>
#include <stdio.h>
#include "midas.h"
main()
{
 INT status, i;
 HNDLE hbuf;
 char event[1000];
 status = cm_connect_experiment("", "Sample", "Producer", NULL);
 if (status != CM_SUCCESS)
 return 1;
 bm_open_buffer(EVENT_BUFFER_NAME, 2*MAX_EVENT_SIZE, &hbuf);

 // create event with ID 1, trigger mask 0, size 100 bytes and serial number 1
 bm_compose_event((EVENT_HEADER *) event, 1, 0, 100, 1);

 // set event data
 for (i=0 ; i<100 ; i++)
 *(event+sizeof(EVENT_HEADER)+i) = i;
 // send event
 bm_send_event(hbuf, event, 100+sizeof(EVENT_HEADER), SYNC);
 cm_disconnect_experiment();
 return 0;
}
```

Parameters:

buffer_handle Buffer handle obtained via bm_open_buffer()

source Address of event buffer

buf_size Size of event including event header in bytes

async_flag Synchronous/asynchronous flag. If FALSE, the function blocks if the buffer has not enough free space to receive the event. If TRUE, the function returns immediately with a value of BM_ASYNC_RETURN without writing the event to the buffer

Returns:

BM_SUCCESS, BM_INVALID_HANDLE, BM_INVALID_PARAM
BM_ASYNC_RETURN Routine called with async_flag == TRUE and buffer has not enough space to receive event
BM_NO_MEMORY Event is too large for network buffer or event buffer. One has to increase MAX_EVENT_SIZE in midas.h and recompile.

Definition at line 5874 of file midas.c.

Referenced by cm_msg(), cm_msg1(), and rpc_send_event().

2.29.1.20 INT bm_set_cache_size (INT *buffer_handle*, INT *read_size*, INT *write_size*)

Modifies buffer cache size. Without a buffer cache, events are copied to/from the shared memory event by event.

To protect processed from accessing the shared memory simultaneously, semaphores are used. Since semaphore operations are CPU consuming (typically 50-100us) this can slow down the data transfer especially for small events. By using a cache the number of semaphore operations is reduced dramatically. Instead writing directly to the shared memory, the events are copied to a local cache buffer. When this buffer is full, it is copied to the shared memory in one operation. The same technique can be used when receiving events.

The drawback of this method is that the events have to be copied twice, once to the cache and once from the cache to the shared memory. Therefore it can happen that the usage of a cache even slows down data throughput on a given environment (computer type, OS type, event size). The cache size has therefore be optimized manually to maximize data throughput.

Parameters:

buffer_handle buffer handle obtained via bm_open_buffer()

read_size cache size for reading events in bytes, zero for no cache

write_size cache size for writing events in bytes, zero for no cache

Returns:

BM_SUCCESS, BM_INVALID_HANDLE, BM_NO_MEMORY, BM_INVALID_PARAM

Definition at line 5007 of file midas.c.

Referenced by register_equipment().

2.29.1.21 INT bm_skip_event (INT *buffer_handle*)

Skip all events in current buffer.

Useful for single event displays to see the newest events

Parameters:

buffer_handle Handle of the buffer. Must be obtained via bm_open_buffer.

Returns:

BM_SUCCESS, BM_INVALID_HANDLE, RPC_NET_ERROR

Definition at line 6538 of file midas.c.

2.29.1.22 BOOL bm_update_read_pointer (const char ∗ *caller_name*, BUFFER_HEADER ∗ *pheader*) `[static]`

Definition at line 5489 of file midas.c.

Referenced by bm_flush_cache(), bm_push_event(), bm_receive_event(), bm_send_event(), and bm_wait_for_free_space().

2.29.1.23 void bm_validate_client_pointers (BUFFER_HEADER ∗ *pheader*, BUFFER_CLIENT ∗ *pclient*) `[static]`

Definition at line 5428 of file midas.c.

Referenced by bm_update_read_pointer().

2.29.1.24 int bm_wait_for_free_space (int *buffer_handle*, BUFFER ∗ *pbuf*, int *async_flag*, int *requested_space*) `[static]`

Definition at line 5667 of file midas.c.

Referenced by bm_flush_cache(), and bm_send_event().

2.29.1.25 void bm_wakeup_producers (const BUFFER_HEADER ∗ *pheader*, const BUFFER_CLIENT ∗ *pc*) `[static]`

Definition at line 5545 of file midas.c.

Referenced by bm_push_event(), and bm_receive_event().

2.30 Midas Message Functions (msg_xxx)

Functions

- INT cm_get_error (INT code, char ∗string)
- INT cm_set_msg_print (INT system_mask, INT user_mask, int(∗func)(const char ∗))
- INT cm_msg_log (INT message_type, const char ∗message)
- INT cm_msg_log1 (INT message_type, const char ∗message, const char ∗facility)
- INT cm_msg (INT message_type, const char ∗filename, INT line, const char ∗routine, const char ∗format,...)
- INT cm_msg1 (INT message_type, const char ∗filename, INT line, const char ∗facility, const char ∗routine, const char ∗format,...)
- INT cm_msg_register (void(∗func)(HNDLE, HNDLE, EVENT_HEADER ∗, void ∗))
- INT cm_msg_retrieve (INT n_message, char ∗message, INT buf_size)

2.30.1 Function Documentation

2.30.1.1 INT cm_get_error (INT *code*, char ∗ *string*)

Convert error code to string. Used after cm_connect_experiment to print error string in command line programs or windows programs.

Parameters:

code Error code as defined in midas.h

string Error string

Returns:

CM_SUCCESS

Definition at line 291 of file midas.c.

Referenced by cm_connect_experiment().

2.30.1.2 INT cm_msg (INT *message_type*, const char ∗ *filename*, INT *line*, const char ∗ *routine*, const char ∗ *format*, ...)

This routine can be called whenever an internal error occurs or an informative message is produced. Different message types can be enabled or disabled by setting the type bits via cm_set_msg_print().

Attention:

Do not add the "\n" escape carriage control at the end of the formated line as it is already added by the client on the receiving side.

```
...
cm_msg(MINFO, "my program", "This is a information message only);
cm_msg(MERROR, "my program", "This is an error message with status:%d", my_status);
cm_msg(MTALK, "my_program", My program is Done!");
...
```

Parameters:

message_type (See MIDAS Macros).

filename Name of source file where error occured

line Line number where error occured

routine Routine name.

format message to printout, ... Parameters like for printf()

Returns:

CM_SUCCESS

Definition at line 554 of file midas.c.

Referenced by al_check(), al_reset_alarm(), al_trigger_alarm(), analyzer_init(), bk_list(), bm_close_buffer(), bm_copy_from_cache(), bm_flush_cache(), bm_open_buffer(), bm_push_event(), bm_receive_event(), bm_remove_event_request(), bm_request_event(), bm_send_event(), bm_set_cache_size(), bm_skip_event(), bm_update_read_pointer(), bm_validate_client_index(), bm_validate_client_pointers(), bm_wait_for_free_space(), bm_wakeup_producers(), close_buffers(), cm_check_client(), cm_check_deferred_transition(), cm_cleanup(), cm_connect_experiment1(), cm_deregister_transition(), cm_disconnect_experiment(), cm_get_watchdog_info(), cm_list_experiments(), cm_register_deferred_transition(), cm_register_transition(), cm_set_client_info(), cm_set_transition_sequence(), cm_shutdown(), cm_transition(), db_check_record(), db_close_database(), db_copy(), db_copy_xml(), db_create_key(), db_create_link(), db_create_record(), db_delete_key1(), db_enum_key(), db_find_key(), db_get_data(), db_get_data_index(), db_get_key(), db_get_key_info(), db_get_key_time(), db_get_record(), db_get_value(), db_load(), db_lock_database(), db_open_database(), db_open_record(), db_paste(), db_paste_node(), db_protect_database(), db_save(), db_save_struct(), db_save_xml(), db_save_xml_key(), db_set_data(), db_set_data_index(), db_set_record(), db_set_value(), db_unlock_database(), dm_buffer_create(), el_submit(), handFlush(), hs_dump(), load_fragment(), main(), readout_thread(), receive_trigger_event(), register_equipment(), rpc_flush_event(), rpc_register_functions(), rpc_send_event(), rpc_set_option(), scan_fragment(), scheduler(), send_event(), source_booking(), source_scan(), source_unbooking(), tr_start(), tr_stop(), update_odb(), and ybk_list().

2.30.1.3 INT cm_msg1 (INT *message_type*, const char ∗ *filename*, INT *line*, const char ∗ *facility*, const char ∗ *routine*, const char ∗ *format*, ...)

This routine is similar to cm_msg(). It differs from cm_msg() only by the logging destination being a file given through the argument list i.e:**facility**

For internal use only.

Attention:

Do not add the "\n" escape carriage control at the end of the formated line as it is already added by the client on the receiving side. The first arg in the following example uses the predefined macro MINFO which handles automatically the first 3 arguments of the function (see MIDAS Macros).

```
    ...
    cm_msg1(MINFO, "my_log_file", "my_program"," My message status:%d", status);
    ...
//----- File my_log_file.log
Thu Nov  8 17:59:28 2001 [my_program] My message status:1
```

Parameters:

message_type	See MIDAS Macros.
filename	Name of source file where error occured
line	Line number where error occured
facility	Logging file name

routine Routine name

format message to printout, ... Parameters like for printf()

Returns:

CM_SUCCESS

Definition at line 671 of file midas.c.

2.30.1.4 INT cm_msg_log (INT *message_type*, const char ∗ *message*)

Write message to logging file. Called by cm_msg.

Attention:

May burn your fingers

Parameters:

message_type Message type

message Message string

Returns:

CM_SUCCESS

Definition at line 353 of file midas.c.

Referenced by cm_msg().

2.30.1.5 INT cm_msg_log1 (INT *message_type*, const char ∗ *message*, const char ∗ *facility*)

Write message to logging file. Called by cm_msg().

For Internal use only.

Parameters:

message_type Message type

message Message string

facility Message facility, filename in which messages will be written

Returns:

CM_SUCCESS

Definition at line 425 of file midas.c.

Referenced by cm_msg1().

2.30.1.6 INT cm_msg_register (void(∗)(HNDLE, HNDLE, EVENT_HEADER ∗, void ∗) *func*)

Register a dispatch function for receiving system messages.

- example code from mlxspeaker.c

```
void receive_message(HNDLE hBuf, HNDLE id, EVENT_HEADER *header, void *message)
{
  char str[256], *pc, *sp;
  // print message
  printf("%s\n", (char *)(message));

  printf("evID:%x Mask:%x Serial:%i Size:%d\n"
                 ,header->event_id
                 ,header->trigger_mask
                 ,header->serial_number
                 ,header->data_size);
  pc = strchr((char *)(message),’]’)+2;
  ...
  // skip none talking message
  if (header->trigger_mask == MT_TALK ||
      header->trigger_mask == MT_USER)
   ...
}

int main(int argc, char *argv[])
{
  ...
  // now connect to server
  status = cm_connect_experiment(host_name, exp_name, "Speaker", NULL);
  if (status != CM_SUCCESS)
    return 1;
  // Register callback for messages
  cm_msg_register(receive_message);
  ...
}
```

Parameters:

func Dispatch function.

Returns:

CM_SUCCESS or bm_open_buffer and bm_request_event return status

Definition at line 803 of file midas.c.

2.30.1.7 INT cm_msg_retrieve (INT *n_message*, char ∗ *message*, INT *buf_size*)

Retrieve old messages from log file

Parameters:

n_message Number of messages to retrieve

message buf_size bytes of messages, separated by

characters. The returned number of bytes is normally smaller than the initial buf_size, since only full lines are returned.

∗***buf_size*** Size of message buffer to fill

Returns:

CM_SUCCESS

Definition at line 833 of file midas.c.

2.30.1.8 INT cm_set_msg_print (INT *system_mask*, INT *user_mask*, int(∗)(const char ∗) *func*)

Set message masks. When a message is generated by calling cm_msg(), it can got to two destinatinons. First a user defined callback routine and second to the "SYSMSG" buffer.

A user defined callback receives all messages which satisfy the user_mask.

```
int message_print(const char *msg)
{
  char str[160];

  memset(str, ' ', 159);
  str[159] = 0;
  if (msg[0] == '[')
    msg = strchr(msg, ']')+2;
  memcpy(str, msg, strlen(msg));
  ss_printf(0, 20, str);
  return 0;
}
...
  cm_set_msg_print(MT_ALL, MT_ALL, message_print);
...
```

Parameters:

system_mask Bit masks for MERROR, MINFO etc. to send system messages.

user_mask Bit masks for MERROR, MINFO etc. to send messages to the user callback.

func Function which receives all printout. By setting "puts", messages are just printed to the screen.

Returns:

CM_SUCCESS

Definition at line 336 of file midas.c.

Referenced by cm_connect_experiment1(), and main().

2.31 Midas Bank Functions (bk_xxx)

Functions

- void bk_init (void ∗event)

- void bk_init32 (void ∗event)
- INT bk_size (void ∗event)
- void bk_create (void ∗event, const char ∗name, WORD type, void ∗pdata)
- INT bk_close (void ∗event, void ∗pdata)
- INT bk_list (void ∗event, char ∗bklist)
- INT bk_locate (void ∗event, const char ∗name, void ∗pdata)
- INT bk_find (BANK_HEADER ∗pbkh, const char ∗name, DWORD ∗bklen, DWORD ∗bktype, void ∗∗pdata)
- INT bk_iterate (void ∗event, BANK ∗∗pbk, void ∗pdata)
- INT bk_swap (void ∗event, BOOL force)

2.31.1 Function Documentation

2.31.1.1 INT bk_close (void ∗ *event*, void ∗ *pdata*)

Close the Midas bank priviously created by bk_create(). The data pointer pdata must be obtained by bk_create() and used as an address to fill a bank. It is incremented with every value written to the bank and finally points to a location just after the last byte of the bank. It is then passed to bk_close() to finish the bank creation

Parameters:

event pointer to current composed event

pdata pointer to the data

Returns:

number of bytes contained in bank

Definition at line 12286 of file midas.c.

Referenced by adc_calib(), adc_summing(), eb_user(), read_scaler_event(), read_trigger_event(), and scaler_accum().

2.31.1.2 void bk_create (void ∗ *event*, const char ∗ *name*, WORD *type*, void ∗ *pdata*)

Create a Midas bank. The data pointer pdata must be used as an address to fill a bank. It is incremented with every value written to the bank and finally points to a location just after the last byte of the bank. It is then passed to the function bk_close() to finish the bank creation.

```
INT *pdata;
bk_init(pevent);
```

```
bk_create(pevent, "ADC0", TID_INT, &pdata);
*pdata++ = 123
*pdata++ = 456
bk_close(pevent, pdata);
```

Parameters:

- ***event*** pointer to the data area
- ***name*** of the bank, must be exactly 4 charaters
- ***type*** type of bank, one of the Midas Data Types values defined in midas.h
- ***pdata*** pointer to the data area of the newly created bank

Returns:

void

Definition at line 12174 of file midas.c.

Referenced by adc_calib(), adc_summing(), eb_user(), read_scaler_event(), read_trigger_event(), and scaler_accum().

2.31.1.3 INT bk_find (BANK_HEADER $*$ *pbkh*, const char $*$ *name*, DWORD $*$ *bklen*, DWORD $*$ *bktype*, void $**$ *pdata*)

Finds a MIDAS bank of given name inside an event.

Parameters:

- ***pbkh*** pointer to current composed event
- ***name*** bank name to look for
- ***bklen*** number of elemtents in bank
- ***bktype*** bank type, one of TID_xxx
- ***pdata*** pointer to data area of bank, NULL if bank not found

Returns:

1 if bank found, 0 otherwise

Definition at line 12430 of file midas.c.

2.31.1.4 void bk_init (void $*$ *event*)

Initializes an event for Midas banks structure. Before banks can be created in an event, bk_init() has to be called first.

Parameters:

- ***event*** pointer to the area of event

Definition at line 12092 of file midas.c.

Referenced by eb_user(), read_scaler_event(), and read_trigger_event().

2.31.1.5 void bk_init32 (void ∗ *event*)

Initializes an event for Midas banks structure for large bank size (> 32KBytes) Before banks can be created in an event, bk_init32() has to be called first.

Parameters:

event pointer to the area of event

Returns:

void

Definition at line 12133 of file midas.c.

2.31.1.6 INT bk_iterate (void ∗ *event*, BANK ∗∗ *pbk*, void ∗ *pdata*)

Iterates through banks inside an event. The function can be used to enumerate all banks of an event. The returned pointer to the bank header has following structure:

```
typedef struct {
char   name[4];
WORD   type;
WORD   data_size;
} BANK;
```

where type is a TID_xxx value and data_size the size of the bank in bytes.

```
BANK *pbk;
INT  size;
void *pdata;
char name[5];
pbk = NULL;
do
{
 size = bk_iterate(event, &pbk, &pdata);
 if (pbk == NULL)
  break;
 *((DWORD *)name) = *((DWORD *)(pbk)->name);
 name[4] = 0;
 printf("bank %s found\n", name);
} while(TRUE);
```

Parameters:

event Pointer to data area of event.

pbk pointer to the bank header, must be NULL for the first call to this function.

pdata Pointer to the bank header, must be NULL for the first call to this function

Returns:

Size of bank in bytes

Definition at line 12511 of file midas.c.

Referenced by bk_list(), and update_odb().

2.31.1.7 INT bk_list (void ∗ *event*, char ∗ *bklist*)

Extract the MIDAS bank name listing of an event. The bklist should be dimensioned with STRING_BANKLIST_MAX which corresponds to a max of BANKLIST_MAX banks (midas.h: 32 banks max).

```
INT adc_calib(EVENT_HEADER *pheader, void *pevent)
{
  INT    n_adc, nbanks;
  WORD   *pdata;
  char   banklist[STRING_BANKLIST_MAX];

  // Display # of banks and list of banks in the event
  nbanks = bk_list(pevent, banklist);
  printf("#banks:%d List:%s\n", nbanks, banklist);

  // look for ADC0 bank, return if not present
  n_adc = bk_locate(pevent, "ADC0", &pdata);
  ...
}
```

Parameters:

event	pointer to current composed event
bklist	returned ASCII string, has to be booked with STRING_BANKLIST_MAX.

Returns:

number of bank found in this event.

Definition at line 12336 of file midas.c.

2.31.1.8 INT bk_locate (void ∗ *event*, const char ∗ *name*, void ∗ *pdata*)

Locates a MIDAS bank of given name inside an event.

Parameters:

event	pointer to current composed event
name	bank name to look for
pdata	pointer to data area of bank, NULL if bank not found

Returns:

number of values inside the bank

Definition at line 12380 of file midas.c.

Referenced by adc_calib(), adc_summing(), and scaler_accum().

2.31.1.9 INT bk_size (void ∗ *event*)

Returns the size of an event containing banks. The total size of an event is the value returned by bk_size() plus the size of the event header (sizeof(EVENT_HEADER)).

Parameters:

event pointer to the area of event

Returns:

number of bytes contained in data area of event

Definition at line 12147 of file midas.c.

Referenced by read_scaler_event(), and read_trigger_event().

2.31.1.10 INT bk_swap (void ∗ *event*, BOOL *force*)

Swaps bytes from little endian to big endian or vice versa for a whole event.

An event contains a flag which is set by bk_init() to identify the endian format of an event. If force is FALSE, this flag is evaluated and the event is only swapped if it is in the "wrong" format for this system. An event can be swapped to the "wrong" format on purpose for example by a front-end which wants to produce events in a "right" format for a back-end analyzer which has different byte ordering.

Parameters:

event pointer to data area of event

force If TRUE, the event is always swapped, if FALSE, the event is only swapped if it is in the wrong format.

Returns:

1==event has been swap, 0==event has not been swapped.

Definition at line 12586 of file midas.c.

Referenced by eb_mfragment_add(), and source_scan().

2.32 Midas RPC Functions (rpc_xxx)

Functions

- INT rpc_register_client (char ∗name, RPC_LIST ∗list)
- INT rpc_register_functions (RPC_LIST ∗new_list, INT(∗func)(INT, void ∗∗))
- INT rpc_set_option (HNDLE hConn, INT item, INT value)
- INT rpc_send_event (INT buffer_handle, void ∗source, INT buf_size, INT async_flag, INT mode)
- INT rpc_flush_event ()

2.32.1 Function Documentation

2.32.1.1 INT rpc_flush_event ()

Send event residing in the TCP cache buffer filled by rpc_send_event. This routine should be called when a run is stopped.

Returns:

RPC_SUCCESS, RPC_NET_ERROR

Definition at line 9718 of file midas.c.

Referenced by scan_fragment(), scheduler(), send_event(), and tr_stop().

2.32.1.2 INT rpc_register_client (char ∗ *name*, RPC_LIST ∗ *list*)

Register RPC client for standalone mode (without standard midas server)

Parameters:

list Array of RPC_LIST structures containing function IDs and parameter definitions. The end of the list must be indicated by a function ID of zero.

name Name of this client

Returns:

RPC_SUCCESS

Definition at line 7571 of file midas.c.

2.32.1.3 INT rpc_register_functions (RPC_LIST ∗ *new_list*, INT(∗)(INT, void ∗∗) *func*)

Register a set of RPC functions (both as clients or servers)

Parameters:

new_list Array of RPC_LIST structures containing function IDs and parameter definitions. The end of the list must be indicated by a function ID of zero.

func Default dispatch function

Returns:

RPC_SUCCESS, RPC_NO_MEMORY, RPC_DOUBLE_DEFINED

Definition at line 7591 of file midas.c.

Referenced by cm_connect_experiment1(), and rpc_register_client().

2.32.1.4 INT rpc_send_event (INT *buffer_handle*, void ∗ *source*, INT *buf_size*, INT *async_flag*, INT *mode*)

Fast send_event routine which bypasses the RPC layer and sends the event directly at the TCP level.

Parameters:

buffer_handle Handle of the buffer to send the event to. Must be obtained via bm_open_buffer.

source Address of the event to send. It must have a proper event header.

buf_size Size of event in bytes with header.

async_flag SYNC / ASYNC flag. In ASYNC mode, the function returns immediately if it cannot send the event over the network. In SYNC mode, it waits until the packet is sent (blocking).

mode Determines in which mode the event is sent. If zero, use RPC socket, if one, use special event socket to bypass RPC layer on the server side.

Returns:

BM_INVALID_PARAM, BM_ASYNC_RETURN, RPC_SUCCESS, RPC_NET_ERROR, RPC_NO_CONNECTION, RPC_EXCEED_BUFFER

Definition at line 9519 of file midas.c.

Referenced by receive_trigger_event(), scheduler(), send_event(), and source_scan().

2.32.1.5 INT rpc_set_option (HNDLE *hConn*, INT *item*, INT *value*)

Set RPC option

Parameters:

hConn RPC connection handle

item One of RPC_Oxxx

value Value to set

Returns:

RPC_SUCCESS

Definition at line 8590 of file midas.c.

Referenced by bm_receive_event(), cm_transition(), db_send_changed_records(), main(), scheduler(), and update_odb().

2.33 Midas Dual Buffer Memory Functions (dm_xxx)

Functions

- INT dm_buffer_create (INT size, INT user_max_event_size)

2.33.1 Function Documentation

2.33.1.1 INT dm_buffer_create (INT *size*, INT *user_max_event_size*)

Setup a dual memory buffer. Has to be called initially before any other dm_xxx function

Parameters:

size Size in bytes

user_max_event_size max event size

Returns:

CM_SUCCESS, BM_NO_MEMORY, BM_MEMSIZE_MISMATCH

Definition at line 13117 of file midas.c.

2.34 Midas Ring Buffer Functions (rb_xxx)

Functions

- int rb_set_nonblocking ()
- int rb_create (int size, int max_event_size, int ∗handle)
- int rb_delete (int handle)
- int rb_get_wp (int handle, void ∗∗p, int millisec)
- int rb_increment_wp (int handle, int size)
- int rb_get_rp (int handle, void ∗∗p, int millisec)
- int rb_increment_rp (int handle, int size)
- int rb_get_buffer_level (int handle, int ∗n_bytes)

2.34.1 Function Documentation

2.34.1.1 int rb_create (int *size*, int *max_event_size*, int ∗ *handle*)

Create a ring buffer with a given size

Provide an inter-thread buffer scheme for handling front-end events. This code allows concurrent data acquisition, calibration and network transfer on a multi-CPU machine. One thread reads out the data, passes it via the ring buffer functions to another thread running on the other CPU, which can then calibrate and/or send the data over the network.

Parameters:

size Size of ring buffer, must be larger than 2∗max_event_size

max_event_size Maximum event size to be placed into

∗***handle*** Handle to ring buffer

Returns:

DB_SUCCESS, DB_NO_MEMORY, DB_INVALID_PARAM

Definition at line 13844 of file midas.c.

Referenced by register_equipment().

2.34.1.2 int rb_delete (int *handle*)

Delete a ring buffer

Parameters:

handle Handle of the ring buffer

Returns:

DB_SUCCESS

Definition at line 13898 of file midas.c.

2.34.1.3 int rb_get_buffer_level (int *handle*, int ∗ *n_bytes*)

Return number of bytes in a ring buffer

Parameters:

handle Handle of the buffer to get the info

∗***n_bytes*** Number of bytes in buffer

Returns:

DB_SUCCESS, DB_INVALID_HANDLE

Definition at line 14188 of file midas.c.

2.34.1.4 int rb_get_rp (int *handle*, void ∗∗ *p*, int *millisec*)

Obtain the current read pointer at which new data is available with optional timeout

Parameters:

handle Ring buffer handle

millisec Optional timeout in milliseconds if buffer is full. Zero to not wait at all (non-blocking)

∗∗***p*** Address of pointer pointing to newly available data. If p == NULL, only return status.

Returns:
DB_SUCCESS, DB_TIEMOUT, DB_INVALID_HANDLE

Definition at line 14072 of file midas.c.

Referenced by receive_trigger_event().

2.34.1.5 int rb_get_wp (int *handle*, void ∗∗ *p*, int *millisec*)

Retrieve write pointer where new data can be written

Parameters:

handle Ring buffer handle

millisec Optional timeout in milliseconds if buffer is full. Zero to not wait at all (non-blocking)

∗∗***p*** Write pointer

Returns:
DB_SUCCESS, DB_TIMEOUT, DB_INVALID_HANDLE

Definition at line 13934 of file midas.c.

Referenced by interrupt_routine(), and readout_thread().

2.34.1.6 int rb_increment_rp (int *handle*, int *size*)

Increment current read pointer, freeing up space for the writing thread.

Parameters:

handle Ring buffer handle

size Number of bytes to free up at current read pointer

Returns:
DB_SUCCESS, DB_INVALID_PARAM

Definition at line 14134 of file midas.c.

Referenced by receive_trigger_event().

2.34.1.7 int rb_increment_wp (int *handle*, int *size*)

rb_increment_wp

Increment current write pointer, making the data at the write pointer available to the receiving thread

Parameters:

handle Ring buffer handle

size Number of bytes placed at the WP

Returns:

DB_SUCCESS, DB_INVALID_PARAM, DB_INVALID_HANDLE

Definition at line 14010 of file midas.c.

Referenced by interrupt_routine(), and readout_thread().

2.34.1.8 int rb_set_nonblocking ()

Set all rb_get_xx to nonblocking. Needed in multi-thread environments for stopping all theads without deadlock

Returns:

DB_SUCCESS

Definition at line 13803 of file midas.c.

Referenced by main().

2.35 System Functions (ss_xxx)

Functions

- INT ss_system (char ∗command)
- midas_thread_t ss_thread_create (INT(∗thread_func)(void ∗), void ∗param)
- INT ss_thread_kill (midas_thread_t thread_id)
- DWORD ss_millitime ()
- DWORD ss_time ()
- INT ss_sleep (INT millisec)

2.35.1 Function Documentation

2.35.1.1 DWORD ss_millitime ()

Returns the actual time in milliseconds with an arbitrary origin. This time may only be used to calculate relative times.

Overruns in the 32 bit value don't hurt since in a subtraction calculated with 32 bit accuracy this overrun cancels (you may think about!)..

```
...
DWORD start, stop:
start = ss_millitime();
  < do operations >
stop = ss_millitime();
printf("Operation took %1.3lf seconds\n",(stop-start)/1000.0);
...
```

Returns:
millisecond time stamp.

Definition at line 2190 of file system.c.

Referenced by bm_check_buffers(), bm_open_buffer(), close_buffers(), cm_cleanup(), cm_get_watchdog_info(), cm_set_watchdog_params(), cm_shutdown(), db_open_database(), dm_buffer_create(), register_equipment(), sc_thread(), scan_fragment(), scheduler(), send_event(), and tr_stop().

2.35.1.2 INT ss_sleep (INT *millisec*)

Suspend the calling process for a certain time.

The function is similar to the sleep() function, but has a resolution of one milliseconds. Under VxWorks the resolution is 1/60 of a second. It uses the socket select() function with a time-out. See examples in ss_time()

Parameters:
millisec Time in milliseconds to sleep. Zero means infinite (until another process calls ss_wake)

Returns:
SS_SUCCESS

Definition at line 2422 of file system.c.

Referenced by cm_shutdown(), device_driver(), main(), rb_get_rp(), rb_get_wp(), read_trigger_event(), readout_thread(), and register_equipment().

2.35.1.3 INT ss_system (char ∗ *command*)

Execute command in a separate process, close all open file descriptors invoke ss_exec() and ignore pid.

```
{ ...
  char cmd[256];
  sprintf(cmd,"%s %s %i %s/%s %1.3lf %d",lazy.commandAfter,
     lazy.backlabel, lazyst.nfiles, lazy.path, lazyst.backfile,
     lazyst.file_size/1024.0/1024.0, blockn);
  cm_msg(MINFO,"Lazy","Exec post file write script:%s",cmd);
  ss_system(cmd);
}
...
```

Parameters:

command Command to execute.

Returns:

SS_SUCCESS or ss_exec() return code

Definition at line 1503 of file system.c.

Referenced by al_check(), and cm_transition().

2.35.1.4 midas_thread_t ss_thread_create (INT(∗)(void ∗) *thread_func*, void ∗ *param*)

Creates and returns a new thread of execution.

Note the difference when calling from vxWorks versus Linux and Windows. The parameter pointer for a vxWorks call is a VX_TASK_SPAWN structure, whereas for Linux and Windows it is a void pointer. Early versions returned SS_SUCCESS or SS_NO_THREAD instead of thread ID.

Example for VxWorks

```
...
VX_TASK_SPAWN tsWatch = {"Watchdog", 100, 0, 2000,  (int) pDevice, 0, 0, 0, 0, 0, 0, 0, 0 ,0};
midas_thread_t thread_id = ss_thread_create((void *) taskWatch, &tsWatch);
if (thread_id == 0) {
  printf("cannot spawn taskWatch\n");
}
...
```

Example for Linux

```
...
midas_thread_t thread_id = ss_thread_create((void *) taskWatch, pDevice);
if (thread_id == 0) {
  printf("cannot spawn taskWatch\n");
}
...
```

Parameters:

(∗thread_func) Thread function to create.

param a pointer to a VX_TASK_SPAWN structure for vxWorks and a void pointer for Unix and Windows

Returns:

the new thread id or zero on error

Definition at line 1629 of file system.c.

Referenced by device_driver(), dm_buffer_create(), and register_equipment().

2.35.1.5 INT ss_thread_kill (midas_thread_t *thread_id*)

Destroys the thread identified by the passed thread id. The thread id is returned by ss_thread_create() on creation.

```
...
midas_thread_t thread_id = ss_thread_create((void *) taskWatch, pDevice);
if (thread_id == 0) {
  printf("cannot spawn taskWatch\n");
}
...
ss_thread_kill(thread_id);
...
```

Parameters:

thread_id the thread id of the thread to be killed.

Returns:

SS_SUCCESS if no error, else SS_NO_THREAD

Definition at line 1703 of file system.c.

Referenced by device_driver().

2.35.1.6 DWORD ss_time ()

Returns the actual time in seconds since 1.1.1970 UTC.

```
...
DWORD start, stop:
start = ss_time();
  ss_sleep(12000);
stop = ss_time();
printf("Operation took %1.3lf seconds\n",stop-start);
...
```

Returns:

Time in seconds

Definition at line 2257 of file system.c.

Referenced by al_check(), al_trigger_alarm(), bm_compose_event(), cm_synchronize(), cm_time(), cm_yield(), db_get_key_time(), db_set_data(), db_set_data_index(), db_set_value(), scheduler(), and send_event().

2.36 The msystem.h & system.c

Modules

- group System Functions (ss_xxx)
- group System Define
- group System Macros
- group System Structure Declaration

2.37 System Define

Defines

- #define DRI_16 (1<<0)
- #define DRI_32 (1<<1)
- #define DRI_64 (1<<2)
- #define DRI_LITTLE_ENDIAN (1<<3)
- #define DRI_BIG_ENDIAN (1<<4)
- #define DRF_IEEE (1<<5)
- #define DRF_G_FLOAT (1<<6)
- #define DR_ASCII (1<<7)

2.37.1 Define Documentation

2.37.1.1 #define DR_ASCII (1<<7)

-

Definition at line 55 of file msystem.h.

2.37.1.2 #define DRF_G_FLOAT (1<<6)

-

Definition at line 54 of file msystem.h.

2.37.1.3 #define DRF_IEEE (1<<5)

-

Definition at line 53 of file msystem.h.

2.37.1.4 #define DRI_16 (1<<0)

-

Definition at line 48 of file msystem.h.

2.37.1.5 #define DRI_32 (1<<1)

-

Definition at line 49 of file msystem.h.

2.37.1.6 #define DRI_64 (1<<2)

-

Definition at line 50 of file msystem.h.

2.37.1.7 #define DRI_BIG_ENDIAN (1<<4)

-

Definition at line 52 of file msystem.h.

2.37.1.8 #define DRI_LITTLE_ENDIAN (1<<3)

-

Definition at line 51 of file msystem.h.

2.38 System Macros

Defines

- #define WORD_SWAP(x)
- #define DWORD_SWAP(x)
- #define QWORD_SWAP(x)

2.38.1 Define Documentation

2.38.1.1 #define DWORD_SWAP(x)

Value:

```
{ BYTE _tmp;                              \
                       _tmp= *((BYTE *)(x));                    \
                       *((BYTE *)(x)) = *(((BYTE *)(x))+3);     \
                       *(((BYTE *)(x))+3) = _tmp;               \
                       _tmp= *(((BYTE *)(x))+1);                \
                       *(((BYTE *)(x))+1) = *(((BYTE *)(x))+2); \
                       *(((BYTE *)(x))+2) = _tmp; }
```

SWAP DWORD macro

Definition at line 75 of file msystem.h.

Referenced by bk_swap().

2.38.1.2 #define QWORD_SWAP(x)

Value:

```
{ BYTE _tmp;                              \
                       _tmp= *((BYTE *)(x));                    \
                       *((BYTE *)(x)) = *(((BYTE *)(x))+7);     \
                       *(((BYTE *)(x))+7) = _tmp;               \
                       _tmp= *(((BYTE *)(x))+1);                \
                       *(((BYTE *)(x))+1) = *(((BYTE *)(x))+6); \
                       *(((BYTE *)(x))+6) = _tmp;               \
                       _tmp= *(((BYTE *)(x))+2);                \
                       *(((BYTE *)(x))+2) = *(((BYTE *)(x))+5); \
                       *(((BYTE *)(x))+5) = _tmp;               \
                       _tmp= *(((BYTE *)(x))+3);                \
                       *(((BYTE *)(x))+3) = *(((BYTE *)(x))+4); \
                       *(((BYTE *)(x))+4) = _tmp; }
```

SWAP QWORD macro

Definition at line 85 of file msystem.h.

Referenced by bk_swap().

2.38.1.3 #define WORD_SWAP(x)

Value:

```
{ BYTE _tmp;                               \
                       _tmp= *((BYTE *)(x));                    \
                       *((BYTE *)(x)) = *(((BYTE *)(x))+1);     \
                       *(((BYTE *)(x))+1) = _tmp; }
```

SWAP WORD macro

Definition at line 68 of file msystem.h.

Referenced by bk_swap().

2.39 System Structure Declaration

Data Structures

- struct FREE_DESCRIP
- struct OPEN_RECORD
- struct DATABASE_CLIENT
- struct DATABASE_HEADER
- struct DATABASE
- struct RECORD_LIST
- struct REQUEST_LIST

2.40 The mrpc.h & mrpc.c

Modules

- group RPC Define
- group Midas RPC_LIST

2.41 RPC Define

Defines

- #define RPC_CM_SET_CLIENT_INFO 11000
- #define RPC_CM_SET_WATCHDOG_PARAMS 11001
- #define RPC_CM_CLEANUP 11002
- #define RPC_CM_GET_WATCHDOG_INFO 11003
- #define RPC_CM_MSG_LOG 11004
- #define RPC_CM_EXECUTE 11005
- #define RPC_CM_SYNCHRONIZE 11006
- #define RPC_CM_ASCTIME 11007
- #define RPC_CM_TIME 11008
- #define RPC_CM_MSG 11009
- #define RPC_CM_EXIST 11011
- #define RPC_CM_MSG_RETRIEVE 11012
- #define RPC_CM_MSG_LOG1 11013
- #define RPC_BM_OPEN_BUFFER 11100
- #define RPC_BM_CLOSE_BUFFER 11101
- #define RPC_BM_CLOSE_ALL_BUFFERS 11102
- #define RPC_BM_GET_BUFFER_INFO 11103
- #define RPC_BM_GET_BUFFER_LEVEL 11104
- #define RPC_BM_INIT_BUFFER_COUNTERS 11105
- #define RPC_BM_SET_CACHE_SIZE 11106
- #define RPC_BM_ADD_EVENT_REQUEST 11107
- #define RPC_BM_REMOVE_EVENT_REQUEST 11108
- #define RPC_BM_SEND_EVENT 11109
- #define RPC_BM_FLUSH_CACHE 11110
- #define RPC_BM_RECEIVE_EVENT 11111
- #define RPC_BM_MARK_READ_WAITING 11112
- #define RPC_BM_EMPTY_BUFFERS 11113
- #define RPC_BM_SKIP_EVENT 11114
- #define RPC_DB_OPEN_DATABASE 11200
- #define RPC_DB_CLOSE_DATABASE 11201
- #define RPC_DB_CLOSE_ALL_DATABASES 11202
- #define RPC_DB_CREATE_KEY 11203
- #define RPC_DB_CREATE_LINK 11204
- #define RPC_DB_SET_VALUE 11205
- #define RPC_DB_GET_VALUE 11206
- #define RPC_DB_FIND_KEY 11207
- #define RPC_DB_FIND_LINK 11208
- #define RPC_DB_GET_PATH 11209
- #define RPC_DB_DELETE_KEY 11210

- #define RPC_DB_ENUM_KEY 11211
- #define RPC_DB_GET_KEY 11212
- #define RPC_DB_GET_DATA 11213
- #define RPC_DB_SET_DATA 11214
- #define RPC_DB_SET_DATA_INDEX 11215
- #define RPC_DB_SET_MODE 11216
- #define RPC_DB_GET_RECORD_SIZE 11219
- #define RPC_DB_GET_RECORD 11220
- #define RPC_DB_SET_RECORD 11221
- #define RPC_DB_ADD_OPEN_RECORD 11222
- #define RPC_DB_REMOVE_OPEN_RECORD 11223
- #define RPC_DB_SAVE 11224
- #define RPC_DB_LOAD 11225
- #define RPC_DB_SET_CLIENT_NAME 11226
- #define RPC_DB_RENAME_KEY 11227
- #define RPC_DB_ENUM_LINK 11228
- #define RPC_DB_REORDER_KEY 11229
- #define RPC_DB_CREATE_RECORD 11230
- #define RPC_DB_GET_DATA_INDEX 11231
- #define RPC_DB_GET_KEY_TIME 11232
- #define RPC_DB_GET_OPEN_RECORDS 11233
- #define RPC_DB_FLUSH_DATABASE 11235
- #define RPC_DB_SET_DATA_INDEX2 11236
- #define RPC_DB_GET_KEY_INFO 11237
- #define RPC_DB_GET_DATA1 11238
- #define RPC_DB_SET_NUM_VALUES 11239
- #define RPC_DB_CHECK_RECORD 11240
- #define RPC_DB_GET_NEXT_LINK 11241
- #define RPC_HS_SET_PATH 11300
- #define RPC_HS_DEFINE_EVENT 11301
- #define RPC_HS_WRITE_EVENT 11302
- #define RPC_HS_COUNT_EVENTS 11303
- #define RPC_HS_ENUM_EVENTS 11304
- #define RPC_HS_COUNT_VARS 11305
- #define RPC_HS_ENUM_VARS 11306
- #define RPC_HS_READ 11307
- #define RPC_HS_GET_VAR 11308
- #define RPC_HS_GET_EVENT_ID 11309
- #define RPC_EL_SUBMIT 11400
- #define RPC_AL_CHECK 11500
- #define RPC_AL_TRIGGER_ALARM 11501
- #define RPC_RC_TRANSITION 12000
- #define RPC_ANA_CLEAR_HISTOS 13000

- #define RPC_LOG_REWIND 14000
- #define RPC_TEST 15000
- #define RPC_CNAF16 16000
- #define RPC_CNAF24 16001
- #define RPC_MANUAL_TRIG 17000
- #define RPC_ID_WATCHDOG 99997
- #define RPC_ID_SHUTDOWN 99998
- #define RPC_ID_EXIT 99999

2.41.1 Define Documentation

2.41.1.1 #define RPC_AL_CHECK 11500

-

Definition at line 121 of file mrpc.h.

Referenced by al_check().

2.41.1.2 #define RPC_AL_TRIGGER_ALARM 11501

-

Definition at line 122 of file mrpc.h.

Referenced by al_trigger_alarm().

2.41.1.3 #define RPC_ANA_CLEAR_HISTOS 13000

-

Definition at line 126 of file mrpc.h.

2.41.1.4 #define RPC_BM_ADD_EVENT_REQUEST 11107

-

Definition at line 59 of file mrpc.h.

2.41.1.5 #define RPC_BM_CLOSE_ALL_BUFFERS 11102

-

Definition at line 54 of file mrpc.h.

Referenced by bm_close_all_buffers().

2.41.1.6 #define RPC_BM_CLOSE_BUFFER 11101

-

Definition at line 53 of file mrpc.h.

Referenced by bm_close_buffer().

2.41.1.7 #define RPC_BM_EMPTY_BUFFERS 11113

-

Definition at line 65 of file mrpc.h.

Referenced by bm_empty_buffers().

2.41.1.8 #define RPC_BM_FLUSH_CACHE 11110

-

Definition at line 62 of file mrpc.h.

Referenced by bm_flush_cache().

2.41.1.9 #define RPC_BM_GET_BUFFER_INFO 11103

-

Definition at line 55 of file mrpc.h.

2.41.1.10 #define RPC_BM_GET_BUFFER_LEVEL 11104

-

Definition at line 56 of file mrpc.h.

2.41.1.11 #define RPC_BM_INIT_BUFFER_COUNTERS 11105

-

Definition at line 57 of file mrpc.h.

2.41.1.12 #define RPC_BM_MARK_READ_WAITING 11112

-

Definition at line 64 of file mrpc.h.

2.41.1.13 #define RPC_BM_OPEN_BUFFER 11100

-

Definition at line 52 of file mrpc.h.

Referenced by bm_open_buffer().

2.41.1.14 #define RPC_BM_RECEIVE_EVENT 11111

-

Definition at line 63 of file mrpc.h.

Referenced by bm_receive_event().

2.41.1.15 #define RPC_BM_REMOVE_EVENT_REQUEST 11108

-

Definition at line 60 of file mrpc.h.

Referenced by bm_remove_event_request().

2.41.1.16 #define RPC_BM_SEND_EVENT 11109

-

Definition at line 61 of file mrpc.h.

Referenced by bm_send_event(), and rpc_send_event().

2.41.1.17 #define RPC_BM_SET_CACHE_SIZE 11106

-

Definition at line 58 of file mrpc.h.

Referenced by bm_set_cache_size().

2.41.1.18 #define RPC_BM_SKIP_EVENT 11114

-

Definition at line 66 of file mrpc.h.

Referenced by bm_skip_event().

2.41.1.19 #define RPC_CM_ASCTIME 11007

-

Definition at line 45 of file mrpc.h.

Referenced by cm_asctime().

2.41.1.20 #define RPC_CM_CLEANUP 11002

-

Definition at line 40 of file mrpc.h.

Referenced by cm_cleanup().

2.41.1.21 #define RPC_CM_EXECUTE 11005

-

Definition at line 43 of file mrpc.h.

Referenced by cm_execute().

2.41.1.22 #define RPC_CM_EXIST 11011

-

Definition at line 48 of file mrpc.h.

Referenced by cm_exist().

2.41.1.23 #define RPC_CM_GET_WATCHDOG_INFO 11003

-

Definition at line 41 of file mrpc.h.

Referenced by cm_get_watchdog_info().

2.41.1.24 #define RPC_CM_MSG 11009

-

Definition at line 47 of file mrpc.h.

2.41.1.25 #define RPC_CM_MSG_LOG 11004

-

Definition at line 42 of file mrpc.h.

Referenced by cm_msg_log().

2.41.1.26 #define RPC_CM_MSG_LOG1 11013

-

Definition at line 50 of file mrpc.h.

Referenced by cm_msg_log1().

2.41.1.27 #define RPC_CM_MSG_RETRIEVE 11012

-

Definition at line 49 of file mrpc.h.

Referenced by cm_msg_retrieve().

2.41.1.28 #define RPC_CM_SET_CLIENT_INFO 11000

-

Definition at line 38 of file mrpc.h.

Referenced by cm_set_client_info().

2.41.1.29 #define RPC_CM_SET_WATCHDOG_PARAMS 11001

-

Definition at line 39 of file mrpc.h.

Referenced by cm_set_watchdog_params().

2.41.1.30 #define RPC_CM_SYNCHRONIZE 11006

-

Definition at line 44 of file mrpc.h.

Referenced by cm_synchronize().

2.41.1.31 #define RPC_CM_TIME 11008

-

Definition at line 46 of file mrpc.h.

Referenced by cm_time().

2.41.1.32 #define RPC_CNAF16 16000

-

Definition at line 132 of file mrpc.h.

2.41.1.33 #define RPC_CNAF24 16001

-

Definition at line 133 of file mrpc.h.

2.41.1.34 #define RPC_DB_ADD_OPEN_RECORD 11222

-

Definition at line 88 of file mrpc.h.

2.41.1.35 #define RPC_DB_CHECK_RECORD 11240

-

Definition at line 105 of file mrpc.h.

Referenced by db_check_record().

2.41.1.36 #define RPC_DB_CLOSE_ALL_DATABASES 11202

-

Definition at line 70 of file mrpc.h.

2.41.1.37 #define RPC_DB_CLOSE_DATABASE 11201

-

Definition at line 69 of file mrpc.h.

Referenced by db_close_database().

2.41.1.38 #define RPC_DB_CREATE_KEY 11203

-

Definition at line 71 of file mrpc.h.

Referenced by db_create_key().

2.41.1.39 #define RPC_DB_CREATE_LINK 11204

-

Definition at line 72 of file mrpc.h.

Referenced by db_create_link().

2.41.1.40 #define RPC_DB_CREATE_RECORD 11230

-

Definition at line 96 of file mrpc.h.

Referenced by db_create_record().

2.41.1.41 #define RPC_DB_DELETE_KEY 11210

-

Definition at line 78 of file mrpc.h.

Referenced by db_delete_key().

2.41.1.42 #define RPC_DB_ENUM_KEY 11211

-

Definition at line 79 of file mrpc.h.

Referenced by db_enum_key().

2.41.1.43 #define RPC_DB_ENUM_LINK 11228

-

Definition at line 94 of file mrpc.h.

2.41.1.44 #define RPC_DB_FIND_KEY 11207

-

Definition at line 75 of file mrpc.h.

Referenced by db_find_key().

2.41.1.45 #define RPC_DB_FIND_LINK 11208

-

Definition at line 76 of file mrpc.h.

2.41.1.46 #define RPC_DB_FLUSH_DATABASE 11235

-

Definition at line 100 of file mrpc.h.

2.41.1.47 #define RPC_DB_GET_DATA 11213

-

Definition at line 81 of file mrpc.h.

Referenced by db_get_data().

2.41.1.48 #define RPC_DB_GET_DATA1 11238

-

Definition at line 103 of file mrpc.h.

2.41.1.49 #define RPC_DB_GET_DATA_INDEX 11231

-

Definition at line 97 of file mrpc.h.

Referenced by db_get_data_index().

2.41.1.50 #define RPC_DB_GET_KEY 11212

-

Definition at line 80 of file mrpc.h.

Referenced by db_get_key().

2.41.1.51 #define RPC_DB_GET_KEY_INFO 11237

-

Definition at line 102 of file mrpc.h.

Referenced by db_get_key_info().

2.41.1.52 #define RPC_DB_GET_KEY_TIME 11232

-

Definition at line 98 of file mrpc.h.

Referenced by db_get_key_time().

2.41.1.53 #define RPC_DB_GET_NEXT_LINK 11241

-

Definition at line 106 of file mrpc.h.

2.41.1.54 #define RPC_DB_GET_OPEN_RECORDS 11233

-

Definition at line 99 of file mrpc.h.

2.41.1.55 #define RPC_DB_GET_PATH 11209

-

Definition at line 77 of file mrpc.h.

2.41.1.56 #define RPC_DB_GET_RECORD 11220

-

Definition at line 86 of file mrpc.h.

Referenced by db_get_record().

2.41.1.57 #define RPC_DB_GET_RECORD_SIZE 11219

-

Definition at line 85 of file mrpc.h.

Referenced by db_get_record_size().

2.41.1.58 #define RPC_DB_GET_VALUE 11206

-

Definition at line 74 of file mrpc.h.

Referenced by db_get_value().

2.41.1.59 #define RPC_DB_LOAD 11225

-

Definition at line 91 of file mrpc.h.

Referenced by db_load().

2.41.1.60 #define RPC_DB_OPEN_DATABASE 11200

-

Definition at line 68 of file mrpc.h.

Referenced by db_open_database().

2.41.1.61 #define RPC_DB_REMOVE_OPEN_RECORD 11223

-

Definition at line 89 of file mrpc.h.

2.41.1.62 #define RPC_DB_RENAME_KEY 11227

-

Definition at line 93 of file mrpc.h.

2.41.1.63 #define RPC_DB_REORDER_KEY 11229

-

Definition at line 95 of file mrpc.h.

2.41.1.64 #define RPC_DB_SAVE 11224

-

Definition at line 90 of file mrpc.h.

Referenced by db_save().

2.41.1.65 #define RPC_DB_SET_CLIENT_NAME 11226

-

Definition at line 92 of file mrpc.h.

2.41.1.66 #define RPC_DB_SET_DATA 11214

-

Definition at line 82 of file mrpc.h.

Referenced by db_set_data().

2.41.1.67 #define RPC_DB_SET_DATA_INDEX 11215

-

Definition at line 83 of file mrpc.h.

Referenced by db_set_data_index().

2.41.1.68 #define RPC_DB_SET_DATA_INDEX2 11236

-

Definition at line 101 of file mrpc.h.

2.41.1.69 #define RPC_DB_SET_MODE 11216

-

Definition at line 84 of file mrpc.h.

2.41.1.70 #define RPC_DB_SET_NUM_VALUES 11239

-

Definition at line 104 of file mrpc.h.

2.41.1.71 #define RPC_DB_SET_RECORD 11221

-

Definition at line 87 of file mrpc.h.

Referenced by db_set_record().

2.41.1.72 #define RPC_DB_SET_VALUE 11205

-

Definition at line 73 of file mrpc.h.

Referenced by db_set_value().

2.41.1.73 #define RPC_EL_SUBMIT 11400

-

Definition at line 119 of file mrpc.h.

Referenced by el_submit().

2.41.1.74 #define RPC_HS_COUNT_EVENTS 11303

-

Definition at line 111 of file mrpc.h.

2.41.1.75 #define RPC_HS_COUNT_VARS 11305

-

Definition at line 113 of file mrpc.h.

2.41.1.76 #define RPC_HS_DEFINE_EVENT 11301

-

Definition at line 109 of file mrpc.h.

2.41.1.77 #define RPC_HS_ENUM_EVENTS 11304

-

Definition at line 112 of file mrpc.h.

2.41.1.78 #define RPC_HS_ENUM_VARS 11306

-

Definition at line 114 of file mrpc.h.

2.41.1.79 #define RPC_HS_GET_EVENT_ID 11309

-

Definition at line 117 of file mrpc.h.

2.41.1.80 #define RPC_HS_GET_VAR 11308

-

Definition at line 116 of file mrpc.h.

2.41.1.81 #define RPC_HS_READ 11307

-

Definition at line 115 of file mrpc.h.

2.41.1.82 #define RPC_HS_SET_PATH 11300

-

Definition at line 108 of file mrpc.h.

Referenced by hs_set_path().

2.41.1.83 #define RPC_HS_WRITE_EVENT 11302

-

Definition at line 110 of file mrpc.h.

2.41.1.84 #define RPC_ID_EXIT 99999

-

Definition at line 139 of file mrpc.h.

2.41.1.85 #define RPC_ID_SHUTDOWN 99998

-

Definition at line 138 of file mrpc.h.

2.41.1.86 #define RPC_ID_WATCHDOG 99997

-

Definition at line 137 of file mrpc.h.

2.41.1.87 #define RPC_LOG_REWIND 14000

-

Definition at line 128 of file mrpc.h.

2.41.1.88 #define RPC_MANUAL_TRIG 17000

-

Definition at line 135 of file mrpc.h.

Referenced by register_equipment().

2.41.1.89 #define RPC_RC_TRANSITION 12000

-

Definition at line 124 of file mrpc.h.

Referenced by cm_register_transition(), and cm_transition().

2.41.1.90 #define RPC_TEST 15000

-

Definition at line 130 of file mrpc.h.

2.42 Midas RPC_LIST

Variables

- RPC_LIST rpc_list_library [ ]
- RPC_LIST rpc_list_system [ ]

2.42.1 Function Documentation

2.42.1.1 RPC_LIST$*$ rpc_get_internal_list (INT *flag*)

Definition at line 1187 of file mrpc.c.

Referenced by cm_connect_experiment1(), rpc_register_client(), and rpc_register_functions().

2.42.2 Variable Documentation

2.42.2.1 RPC_LIST rpc_list_library[ ] `[static]`

rpc_list_library contains all MIDAS library functions and gets registerd whenever a connection to the MIDAS server is established

Definition at line 44 of file mrpc.c.

2.42.2.2 RPC_LIST rpc_list_system[ ] `[static]`

Initial value:

```
 {

   {RPC_ID_WATCHDOG, "id_watchdog",
    {{0}}},

   {RPC_ID_SHUTDOWN, "id_shutdown",
    {{0}}},

   {RPC_ID_EXIT, "id_exit",
    {{0}}},

   {0}

}
```

rpc_list_system contains MIDAS system functions and gets registerd whenever a RPC server is registered

Definition at line 1171 of file mrpc.c.

2.43 The odb.c

Modules

- group Midas ODB Functions (db_xxx)

2.44 Midas ODB Functions (db_xxx)

Functions

- INT db_open_database (char ∗database_name, INT database_size, HNDLE ∗hDB, char ∗client_name)
- INT db_close_database (HNDLE hDB)
- INT db_lock_database (HNDLE hDB)
- INT db_unlock_database (HNDLE hDB)
- INT db_protect_database (HNDLE hDB)
- INT db_create_key (HNDLE hDB, HNDLE hKey, char ∗key_name, DWORD type)
- INT db_create_link (HNDLE hDB, HNDLE hKey, char ∗link_name, char ∗destination)

- INT db_delete_key1 (HNDLE hDB, HNDLE hKey, INT level, BOOL follow_links)
- INT db_delete_key (HNDLE hDB, HNDLE hKey, BOOL follow_links)
- INT db_find_key (HNDLE hDB, HNDLE hKey, char ∗key_name, HNDLE ∗subhKey)
- INT db_set_value (HNDLE hDB, HNDLE hKeyRoot, char ∗key_name, void ∗data, INT data_size, INT num_values, DWORD type)
- INT db_set_value_index (HNDLE hDB, HNDLE hKeyRoot, char ∗key_name, void ∗data, INT data_size, INT index, DWORD type, BOOL truncate)
- INT db_get_value (HNDLE hDB, HNDLE hKeyRoot, char ∗key_name, void ∗data, INT ∗buf_size, DWORD type, BOOL create)
- INT db_enum_key (HNDLE hDB, HNDLE hKey, INT idx, HNDLE ∗subkey_handle)
- INT db_get_key (HNDLE hDB, HNDLE hKey, KEY ∗key)
- INT db_get_key_time (HNDLE hDB, HNDLE hKey, DWORD ∗delta)
- INT db_get_key_info (HNDLE hDB, HNDLE hKey, char ∗name, INT name_size, INT ∗type, INT ∗num_values, INT ∗item_size)
- INT db_get_data (HNDLE hDB, HNDLE hKey, void ∗data, INT ∗buf_size, DWORD type)
- INT db_get_data_index (HNDLE hDB, HNDLE hKey, void ∗data, INT ∗buf_size, INT idx, DWORD type)
- INT db_set_data (HNDLE hDB, HNDLE hKey, void ∗data, INT buf_size, INT num_values, DWORD type)
- INT db_set_data_index (HNDLE hDB, HNDLE hKey, void ∗data, INT data_size, INT idx, DWORD type)
- INT db_load (HNDLE hDB, HNDLE hKeyRoot, char ∗filename, BOOL bRemote)
- INT db_copy (HNDLE hDB, HNDLE hKey, char ∗buffer, INT ∗buffer_size, char ∗path)
- INT db_paste (HNDLE hDB, HNDLE hKeyRoot, char ∗buffer)
- INT db_paste_xml (HNDLE hDB, HNDLE hKeyRoot, char ∗buffer)
- INT db_copy_xml (HNDLE hDB, HNDLE hKey, char ∗buffer, INT ∗buffer_size)
- INT db_save (HNDLE hDB, HNDLE hKey, char ∗filename, BOOL bRemote)
- INT db_save_xml (HNDLE hDB, HNDLE hKey, char ∗filename)
- INT db_save_struct (HNDLE hDB, HNDLE hKey, char ∗file_name, char ∗struct_name, BOOL append)
- INT db_sprintf (char ∗string, void ∗data, INT data_size, INT idx, DWORD type)
- INT db_get_record_size (HNDLE hDB, HNDLE hKey, INT align, INT ∗buf_size)
- INT db_get_record (HNDLE hDB, HNDLE hKey, void ∗data, INT ∗buf_size, INT align)
- INT db_set_record (HNDLE hDB, HNDLE hKey, void ∗data, INT buf_size, INT align)

- INT db_create_record (HNDLE hDB, HNDLE hKey, char ∗orig_key_name, char ∗init_str)
- INT db_check_record (HNDLE hDB, HNDLE hKey, char ∗keyname, char ∗rec_str, BOOL correct)
- INT db_open_record (HNDLE hDB, HNDLE hKey, void ∗ptr, INT rec_size, WORD access_mode, void(∗dispatcher)(INT, INT, void ∗), void ∗info)
- INT db_close_record (HNDLE hDB, HNDLE hKey)
- INT db_close_all_records ()
- INT db_update_record (INT hDB, INT hKey, int s)
- INT db_send_changed_records ()

2.44.1 Function Documentation

2.44.1.1 INT db_check_record (HNDLE *hDB*, HNDLE *hKey*, char ∗ *keyname*, char ∗ *rec_str*, BOOL *correct*)

This function ensures that a certain ODB subtree matches a given C structure, by comparing the init_str with the current ODB structure. If the record does not exist at all, it is created with the default values in init_str. If it does exist but does not match the variables in init_str, the function returns an error if correct=FALSE or calls db_create_record() if correct=TRUE.

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKey Handle for key where search starts, zero for root.

keyname Name of key to search, can contain directories.

rec_str ASCII representation of ODB record in the format

correct If TRUE, correct ODB record if necessary

Returns:

DB_SUCCESS, DB_INVALID_HANDLE, DB_NO_KEY, DB_STRUCT_MISMATCH

Definition at line 7528 of file odb.c.

Referenced by al_check(), al_trigger_alarm(), cm_connect_experiment1(), and register_equipment().

2.44.1.2 INT db_close_all_records ()

Release local memory for open records. This routines is called by db_close_all_databases() and cm_disconnect_experiment()

Returns:

DB_SUCCESS, DB_INVALID_HANDLE

Definition at line 8008 of file odb.c.

Referenced by cm_disconnect_experiment().

2.44.1.3 INT db_close_database (HNDLE *hDB*)

Close a database

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

Returns:

DB_SUCCESS, DB_INVALID_HANDLE, RPC_NET_ERROR

Definition at line 1036 of file odb.c.

2.44.1.4 INT db_close_record (HNDLE *hDB*, HNDLE *hKey*)

Close a record previously opend with db_open_record.

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKey Handle for key where search starts, zero for root.

Returns:

DB_SUCCESS, DB_INVALID_HANDLE

Definition at line 7971 of file odb.c.

2.44.1.5 INT db_copy (HNDLE *hDB*, HNDLE *hKey*, char ∗ *buffer*, INT ∗ *buffer_size*, char ∗ *path*)

Copy an ODB subtree in ASCII format to a buffer

This function converts the binary ODB contents to an ASCII. The function db_paste() can be used to convert the ASCII representation back to binary ODB contents. The functions db_load() and db_save() internally use db_copy() and db_paste(). This function converts the binary ODB contents to an ASCII representation of the form:

- For single value:

```
[ODB path]
 key name = type : value
```

- For strings:

```
key name = STRING : [size] string contents
```

- For arrayes (type can be BYTE, SBYTE, CHAR, WORD, SHORT, DWORD, INT, BOOL, FLOAT, DOUBLE, STRING or LINK):

```
key name = type[size] :
 [0] value0
 [1] value1
 [2] value2
 ...
```

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKey Handle for key where search starts, zero for root.

buffer ASCII buffer which receives ODB contents.

buffer_size Size of buffer, returns remaining space in buffer.

path Internal use only, must be empty ("").

Returns:

DB_SUCCESS, DB_TRUNCATED, DB_NO_MEMORY

Definition at line 5050 of file odb.c.

Referenced by db_create_record(), and db_save().

2.44.1.6 INT db_copy_xml (HNDLE *hDB*, HNDLE *hKey*, char ∗ *buffer*, INT ∗ *buffer_size*)

Copy an ODB subtree in XML format to a buffer

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKey Handle for key where search starts, zero for root.

buffer ASCII buffer which receives ODB contents.

buffer_size Size of buffer, returns remaining space in buffer.

Returns:

DB_SUCCESS, DB_TRUNCATED, DB_NO_MEMORY

Definition at line 5759 of file odb.c.

2.44.1.7 INT db_create_key (HNDLE *hDB*, HNDLE *hKey*, char ∗ *key_name*, DWORD *type*)

Create a new key in a database

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKey Key handle to start with, 0 for root

key_name Name of key in the form "/key/key/key"

type Type of key, one of TID_xxx (see Midas Data Types)

Returns:

DB_SUCCESS, DB_INVALID_HANDLE, DB_INVALID_PARAM, DB_FULL, DB_KEY_EXIST, DB_NO_ACCESS

Definition at line 1458 of file odb.c.

Referenced by db_create_record(), db_get_value(), db_paste(), db_paste_node(), db_set_value(), db_set_value_index(), and register_equipment().

2.44.1.8 INT db_create_link (HNDLE *hDB*, HNDLE *hKey*, char ∗ *link_name*, char ∗ *destination*)

Create a link to a key or set the destination of and existing link.

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKey Key handle to start with, 0 for root

link_name Name of key in the form "/key/key/key"

destination Destination of link in the form "/key/key/key"

Returns:

DB_SUCCESS, DB_INVALID_HANDLE, DB_FULL, DB_KEY_EXIST, DB_NO_ACCESS

Definition at line 1697 of file odb.c.

2.44.1.9 INT db_create_record (HNDLE *hDB*, HNDLE *hKey*, char ∗ *orig_key_name*, char ∗ *init_str*)

Create a record. If a part of the record exists alreay, merge it with the init_str (use values from the init_str only when they are not in the existing record).

This functions creates a ODB sub-tree according to an ASCII representation of that tree. See db_copy() for a description. It can be used to create a sub-tree which exactly matches a C structure. The sub-tree can then later mapped to the C structure ("hot-link") via the function db_open_record().

If a sub-tree exists already which exactly matches the ASCII representation, it is not modified. If part of the tree exists, it is merged with the ASCII representation where the ODB values have priority, only values not present in the ODB are created with the default values of the ASCII representation. It is therefore recommended that before creating an ODB hot-link the function db_create_record() is called to insure that the ODB tree and the C structure contain exactly the same values in the same order.

Following example creates a record under /Equipment/Trigger/Settings, opens a hot-link between that record and a local C structure trigger_settings and registers a callback function trigger_update() which gets called each time the record is changed.

```
struct {
  INT level1;
  INT level2;
} trigger_settings;
char *trigger_settings_str =
"[Settings]\n\
level1 = INT : 0\n\
level2 = INT : 0";
void trigger_update(INT hDB, INT hkey, void *info)
{
  printf("New levels: %d %d\n",
    trigger_settings.level1,
    trigger_settings.level2);
}
main()
{
  HNDLE hDB, hkey;
  char[128] info;
  ...
  cm_get_experiment_database(&hDB, NULL);
  db_create_record(hDB, 0, "/Equipment/Trigger", trigger_settings_str);
  db_find_key(hDB, 0,"/Equipment/Trigger/Settings", &hkey);
  db_open_record(hDB, hkey, &trigger_settings,
    sizeof(trigger_settings), MODE_READ, trigger_update, info);
  ...
}
```

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKey Handle for key where search starts, zero for root.

orig_key_name Name of key to search, can contain directories.

init_str Initialization string in the format of the db_copy/db_save functions.

Returns:

DB_SUCCESS, DB_INVALID_HANDLE, DB_FULL, DB_NO_ACCESS, DB_OPEN_RECORD

Definition at line 7358 of file odb.c.

Referenced by al_check(), al_trigger_alarm(), analyzer_init(), cm_set_client_info(), db_check_record(), main(), register_equipment(), and tr_start().

2.44.1.10 INT db_delete_key (HNDLE *hDB*, HNDLE *hKey*, BOOL *follow_links*)

Delete a subtree in a database starting from a key (including this key).

```
...
    status = db_find_link(hDB, 0, str, &hkey);
    if (status != DB_SUCCESS)
    {
      cm_msg(MINFO,"my_delete"," "Cannot find key %s", str);
      return;
    }

    status = db_delete_key(hDB, hkey, FALSE);
    if (status != DB_SUCCESS)
    {
      cm_msg(MERROR,"my_delete"," "Cannot delete key %s", str);
      return;
    }
  ...
```

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKey for key where search starts, zero for root.

follow_links Follow links when TRUE.

Returns:

DB_SUCCESS, DB_INVALID_HANDLE, DB_NO_ACCESS, DB_OPEN_RECORD

Definition at line 1897 of file odb.c.

Referenced by cm_deregister_transition(), cm_set_client_info(), and db_create_record().

2.44.1.11 INT db_delete_key1 (HNDLE *hDB*, HNDLE *hKey*, INT *level*, BOOL *follow_links*)

Delete a subtree, using level information (only called internally by db_delete_key())

For Internal use only.

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKey Key handle to start with, 0 for root

level Recursion level, must be zero when

follow_links Follow links when TRUE called from a user routine

Returns:

DB_SUCCESS, DB_INVALID_HANDLE, DB_OPEN_RECORD

Definition at line 1727 of file odb.c.

Referenced by cm_delete_client_info(), and db_delete_key().

2.44.1.12 INT db_enum_key (HNDLE *hDB*, HNDLE *hKey*, INT *idx*, HNDLE ∗ *subkey_handle*)

Enumerate subkeys from a key, follow links.

hkey must correspond to a valid ODB directory. The index is usually incremented in a loop until the last key is reached. Information about the sub-keys can be obtained with db_get_key(). If a returned key is of type TID_KEY, it contains itself sub-keys. To scan a whole ODB sub-tree, the function db_scan_tree() can be used.

```
INT   i;
HNDLE hkey, hsubkey;
KEY   key;
  db_find_key(hdb, 0, "/Runinfo", &hkey);
  for (i=0 ; ; i++)
  {
   db_enum_key(hdb, hkey, i, &hsubkey);
   if (!hSubkey)
    break; // end of list reached
   // print key name
   db_get_key(hdb, hkey, &key);
   printf("%s\n", key.name);
  }
```

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKey Handle for key where search starts, zero for root.

idx Subkey index, sould be initially 0, then incremented in each call until subhKey becomes zero and the function returns DB_NO_MORE_SUBKEYS

subkey_handle Handle of subkey which can be used in db_get_key() and db_get_data()

Returns:

DB_SUCCESS, DB_INVALID_HANDLE, DB_NO_MORE_SUBKEYS

Definition at line 3208 of file odb.c.

Referenced by al_check(), cm_connect_client(), cm_exist(), cm_set_client_info(), cm_shutdown(), cm_transition(), load_fragment(), logger_root(), and update_odb().

2.44.1.13 INT db_find_key (HNDLE *hDB*, HNDLE *hKey*, char ∗ *key_name*, HNDLE ∗ *subhKey*)

Returns key handle for a key with a specific name.

Keys can be accessed by their name including the directory or by a handle. A key handle is an internal offset to the shared memory where the ODB lives and allows a much faster access to a key than via its name.

The function db_find_key() must be used to convert a key name to a handle. Most other database functions use this key handle in various operations.

```
HNDLE hkey, hsubkey;
// use full name, start from root
db_find_key(hDB, 0, "/Runinfo/Run number", &hkey);
// start from subdirectory
db_find_key(hDB, 0, "/Runinfo", &hkey);
db_find_key(hdb, hkey, "Run number", &hsubkey);
```

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKey Handle for key where search starts, zero for root.

key_name Name of key to search, can contain directories.

subhKey Returned handle of key, zero if key cannot be found.

Returns:

DB_SUCCESS, DB_INVALID_HANDLE, DB_NO_ACCESS, DB_NO_KEY

Definition at line 1930 of file odb.c.

Referenced by al_check(), al_reset_alarm(), al_trigger_alarm(), analyzer_init(), cm_connect_client(), cm_deregister_transition(), cm_exist(), cm_get_client_info(), cm_msg_log(), cm_msg_log1(), cm_msg_retrieve(), cm_register_deferred_transition(), cm_register_transition(), cm_set_client_info(), cm_shutdown(), cm_transition(), db_check_record(), db_create_link(), db_create_record(), db_delete_key1(), db_enum_key(), db_get_value(), db_paste(), db_paste_xml(), db_set_value(), db_set_value_index(), load_fragment(), logger_root(), main(), register_equipment(), tr_start(), and update_odb().

2.44.1.14 INT db_get_data (HNDLE *hDB*, HNDLE *hKey*, void ∗ *data*, INT ∗ *buf_size*, DWORD *type*)

Get key data from a handle

The function returns single values or whole arrays which are contained in an ODB key. Since the data buffer is of type void, no type checking can be performed by the compiler. Therefore the type has to be explicitly supplied, which is checked against the type stored in the ODB.

```
HNLDE hkey;
INT   run_number, size;
// get key handle for run number
db_find_key(hDB, 0, "/Runinfo/Run number", &hkey);
// return run number
size = sizeof(run_number);
db_get_data(hDB, hkey, &run_number, &size,TID_INT);
```

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKey Handle for key where search starts, zero for root.

data Pointer to the return data.

buf_size Size of data buffer.

type Type of key, one of TID_xxx (see Midas Data Types).

Returns:

DB_SUCCESS, DB_INVALID_HANDLE, DB_TRUNCATED, DB_TYPE_MISMATCH

Definition at line 3977 of file odb.c.

Referenced by cm_connect_client(), cm_get_client_info(), cm_set_client_info(), db_copy(), db_get_record(), db_save_xml_key(), and tr_start().

2.44.1.15 INT db_get_data_index (HNDLE *hDB*, HNDLE *hKey*, void ∗ *data*, INT ∗ *buf_size*, INT *idx*, DWORD *type*)

returns a single value of keys containing arrays of values.

The function returns a single value of keys containing arrays of values.

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKey Handle for key where search starts, zero for root.

data Size of data buffer.

buf_size Return size of the record.

idx Index of array [0..n-1].

type Type of key, one of TID_xxx (see Midas Data Types).

Returns:

DB_SUCCESS, DB_INVALID_HANDLE, DB_TRUNCATED, DB_OUT_OF_RANGE

Definition at line 4203 of file odb.c.

Referenced by cm_transition().

2.44.1.16 INT db_get_key (HNDLE *hDB*, HNDLE *hKey*, KEY ∗ *key*)

Get key structure from a handle.

KEY structure has following format:

```
typedef struct {
  DWORD         type;                 // TID_xxx type
  INT           num_values;           // number of values
  char          name[NAME_LENGTH];    // name of variable
  INT           data;                 // Address of variable (offset)
  INT           total_size;           // Total size of data block
  INT           item_size;            // Size of single data item
  WORD          access_mode;          // Access mode
  WORD          notify_count;         // Notify counter
  INT           next_key;             // Address of next key
  INT           parent_keylist;       // keylist to which this key belongs
  INT           last_written;         // Time of last write action
} KEY;
```

Most of these values are used for internal purposes, the values which are of public interest are type, num_values, and name. For keys which contain a single value, num_values equals to one and total_size equals to item_size. For keys which contain an array of strings (TID_STRING), item_size equals to the length of one string.

```
KEY   key;
HNDLE hkey;
db_find_key(hDB, 0, "/Runinfo/Run number", &hkey);
db_get_key(hDB, hkey, &key);
printf("The run number is of type %s\n", rpc_tid_name(key.type));
```

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKey Handle for key where search starts, zero for root.

key Pointer to KEY stucture.

Returns:

DB_SUCCESS, DB_INVALID_HANDLE

Definition at line 3540 of file odb.c.

Referenced by al_check(), al_reset_alarm(), cm_check_client(), cm_register_transition(), cm_shutdown(), cm_transition(), db_check_record(), db_copy(), db_get_record(), db_get_record_size(), db_open_record(), db_paste(), db_save_struct(), db_save_xml_key(), db_set_record(), load_fragment(), tr_start(), and update_odb().

2.44.1.17 INT db_get_key_info (HNDLE *hDB*, HNDLE *hKey*, char ∗ *name*, INT *name_size*, INT ∗ *type*, INT ∗ *num_values*, INT ∗ *item_size*)

Get key info (separate values instead of structure)

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKey Handle of key to operate on

name Key name

name_size Size of the give name (done with sizeof())

type Key type (see Midas Data Types).

num_values Number of values in key.

item_size Size of individual key value (used for strings)

Returns:

DB_SUCCESS, DB_INVALID_HANDLE

Definition at line 3662 of file odb.c.

2.44.1.18 INT db_get_key_time (HNDLE *hDB*, HNDLE *hKey*, DWORD ∗ *delta*)

Get time when key was last modified

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKey Handle of key to operate on

delta Seconds since last update

Returns:

DB_SUCCESS, DB_INVALID_HANDLE

Definition at line 3604 of file odb.c.

2.44.1.19 INT db_get_record (HNDLE *hDB*, HNDLE *hKey*, void ∗ *data*, INT ∗ *buf_size*, INT *align*)

Copy a set of keys to local memory.

An ODB sub-tree can be mapped to a C structure automatically via a hot-link using the function db_open_record() or manually with this function. Problems might occur if the ODB sub-tree contains values which don't match the C structure. Although the structure size is checked against the sub-tree size, no checking can be done if the type and order of the values in the structure are the same than those in the ODB sub-tree. Therefore it is recommended to use the function db_create_record() before db_get_record() is used which ensures that both are equivalent.

```
struct {
  INT level1;
  INT level2;
} trigger_settings;
char *trigger_settings_str =
"[Settings]\n\
level1 = INT : 0\n\
level2 = INT : 0";

main()
{
  HNDLE hDB, hkey;
  INT   size;
  ...
  cm_get_experiment_database(&hDB, NULL);
  db_create_record(hDB, 0, "/Equipment/Trigger", trigger_settings_str);
  db_find_key(hDB, 0, "/Equipment/Trigger/Settings", &hkey);
  size = sizeof(trigger_settings);
  db_get_record(hDB, hkey, &trigger_settings, &size, 0);
  ...
}
```

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKey Handle for key where search starts, zero for root.

data Pointer to the retrieved data.

buf_size Size of data structure, must be obtained via sizeof(RECORD-NAME).

align Byte alignment calculated by the stub and passed to the rpc side to align data according to local machine. Must be zero when called from user level.

Returns:

DB_SUCCESS, DB_INVALID_HANDLE, DB_STRUCT_SIZE_MISMATCH

Definition at line 6847 of file odb.c.

Referenced by al_check(), al_reset_alarm(), al_trigger_alarm(), cm_transition(), db_open_record(), db_update_record(), register_equipment(), and tr_start().

2.44.1.20 INT db_get_record_size (HNDLE *hDB*, HNDLE *hKey*, INT *align*, INT ∗ *buf_size*)

Calculates the size of a record.

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKey Handle for key where search starts, zero for root.

align Byte alignment calculated by the stub and passed to the rpc side to align data according to local machine. Must be zero when called from user level

buf_size Size of record structure

Returns:

DB_SUCCESS, DB_INVALID_HANDLE, DB_TYPE_MISMATCH, DB_STRUCT_SIZE_MISMATCH, DB_NO_KEY

Definition at line 6761 of file odb.c.

Referenced by db_get_record(), db_open_record(), and db_set_record().

2.44.1.21 INT db_get_value (HNDLE *hDB*, HNDLE *hKeyRoot*, char ∗ *key_name*, void ∗ *data*, INT ∗ *buf_size*, DWORD *type*, BOOL *create*)

Get value of a single key.

The function returns single values or whole arrays which are contained in an ODB key. Since the data buffer is of type void, no type checking can be performed by the compiler. Therefore the type has to be explicitly supplied, which is checked against the type stored in the ODB. key_name can contain the full path of a key (like: "/Equipment/Trigger/Settings/Level1") while hkey is zero which refers to the root, or hkey can refer to a sub-directory (like: /Equipment/Trigger) and key_name is interpreted relative to that directory like "Settings/Level1".

```
INT level1, size;
  size = sizeof(level1);
  db_get_value(hDB, 0, "/Equipment/Trigger/Settings/Level1",
                                 &level1, &size, TID_INT, 0);
```

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKeyRoot Handle for key where search starts, zero for root.

key_name Name of key to search, can contain directories.

data Address of data.

buf_size Maximum buffer size on input, number of written bytes on return.

type Type of key, one of TID_xxx (see Midas Data Types)

create If TRUE, create key if not existing

Returns:

DB_SUCCESS, DB_INVALID_HANDLE, DB_NO_ACCESS, DB_TYPE_MISMATCH, DB_TRUNCATED, DB_NO_KEY

Definition at line 3049 of file odb.c.

Referenced by al_check(), al_trigger_alarm(), ana_end_of_run(), bm_open_buffer(), cm_check_client(), cm_connect_experiment1(), cm_exist(), cm_msg_log(), cm_msg_log1(), cm_msg_retrieve(), cm_register_deferred_transition(), cm_set_client_info(), cm_shutdown(), cm_transition(), el_submit(), load_fragment(), logger_root(), main(), register_equipment(), scheduler(), and tr_start().

2.44.1.22 INT db_load (HNDLE *hDB*, HNDLE *hKeyRoot*, char ∗ *filename*, BOOL *bRemote*)

Load a branch of a database from an .ODB file.

This function is used by the ODBEdit command load. For a description of the ASCII format, see db_copy(). Data can be loaded relative to the root of the ODB (hkey equal zero) or relative to a certain key.

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKeyRoot Handle for key where search starts, zero for root.

filename Filename of .ODB file.

bRemote If TRUE, the file is loaded by the server process on the back-end, if FALSE, it is loaded from the current process

Returns:

DB_SUCCESS, DB_INVALID_HANDLE, DB_FILE_ERROR

Definition at line 4965 of file odb.c.

2.44.1.23 INT db_lock_database (HNDLE *hDB*)

Lock a database for exclusive access via system mutex calls.

Parameters:

hDB Handle to the database to lock

Returns:

DB_SUCCESS, DB_INVALID_HANDLE, DB_TIMEOUT

Definition at line 1316 of file odb.c.

Referenced by cm_check_client(), cm_cleanup(), cm_delete_client_info(), cm_get_watchdog_info(), cm_set_client_info(), cm_set_watchdog_params(), db_close_database(), db_create_key(), db_create_record(), db_delete_key1(), db_enum_key(), db_find_key(), db_get_data(), db_get_data_index(), db_get_key(), db_get_key_info(), db_get_key_time(), db_get_record(), db_get_record_size(), db_get_value(), db_open_database(), db_set_data(), db_set_data_index(), db_set_record(), and db_set_value().

2.44.1.24 INT db_open_database (char ∗ *database_name*, INT *database_size*, HNDLE ∗ *hDB*, char ∗ *client_name*)

Open an online database

Parameters:

database_name Database name.

database_size Initial size of database if not existing

client_name Name of this application

hDB ODB handle obtained via cm_get_experiment_database().

Returns:

DB_SUCCESS, DB_CREATED, DB_INVALID_NAME, DB_NO_MEMORY, DB_MEMSIZE_MISMATCH, DB_NO_MUTEX, DB_INVALID_PARAM, RPC_NET_ERROR

Definition at line 723 of file odb.c.

Referenced by cm_connect_experiment1().

2.44.1.25 INT db_open_record (HNDLE *hDB*, HNDLE *hKey*, void $*$ *ptr*, INT *rec_size*, WORD *access_mode*, void($*$)(INT, INT, void $*$) *dispatcher*, void $*$ *info*)

Open a record. Create a local copy and maintain an automatic update.

This function opens a hot-link between an ODB sub-tree and a local structure. The sub-tree is copied to the structure automatically every time it is modified by someone else. Additionally, a callback function can be declared which is called after the structure has been updated. The callback function receives the database handle and the key handle as parameters.

Problems might occur if the ODB sub-tree contains values which don't match the C structure. Although the structure size is checked against the sub-tree size, no checking can be done if the type and order of the values in the structure are the same than those in the ODB sub-tree. Therefore it is recommended to use the function db_create_record() before db_open_record() is used which ensures that both are equivalent.

The access mode might either be MODE_READ or MODE_WRITE. In read mode, the ODB sub-tree is automatically copied to the local structure when modified by other clients. In write mode, the local structure is copied to the ODB sub-tree if it has been modified locally. This update has to be manually scheduled by calling db_send_changed_records() periodically in the main loop. The system keeps a copy of the local structure to determine if its contents has been changed.

If MODE_ALLOC is or'ed with the access mode, the memory for the structure is allocated internally. The structure pointer must contain a pointer to a pointer to the structure. The internal memory is released when db_close_record() is called.

- To open a record in write mode.

```
struct {
  INT level1;
  INT level2;
} trigger_settings;
char *trigger_settings_str =
"[Settings]\n\
level1 = INT : 0\n\
level2 = INT : 0";
main()
{
  HNDLE hDB, hkey, i=0;
  ...
  cm_get_experiment_database(&hDB, NULL);
  db_create_record(hDB, 0, "/Equipment/Trigger", trigger_settings_str);
  db_find_key(hDB, 0,"/Equipment/Trigger/Settings", &hkey);
  db_open_record(hDB, hkey, &trigger_settings, sizeof(trigger_settings)
                , MODE_WRITE, NULL);
  do
  {
    trigger_settings.level1 = i++;
    db_send_changed_records()
    status = cm_yield(1000);
  } while (status != RPC_SHUTDOWN && status != SS_ABORT);
  ...
}
```

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKey Handle for key where search starts, zero for root.

ptr If access_mode includes MODE_ALLOC: Address of pointer which points to the record data after the call if access_mode includes not MODE_ALLOC: Address of record if ptr==NULL, only the dispatcher is called.

rec_size Record size in bytes

access_mode Mode for opening record, either MODE_READ or MODE_WRITE. May be or'ed with MODE_ALLOC to let db_open_record allocate the memory for the record.

(∗dispatcher) Function which gets called when record is updated.The argument list composed of: HNDLE hDB, HNDLE hKey, void ∗info

info Additional info passed to the dispatcher.

Returns:

DB_SUCCESS, DB_INVALID_HANDLE, DB_NO_MEMORY, DB_NO_ACCESS, DB_STRUCT_SIZE_MISMATCH

Definition at line 7837 of file odb.c.

Referenced by analyzer_init(), cm_register_deferred_transition(), and register_equipment().

2.44.1.26 INT db_paste (HNDLE *hDB*, HNDLE *hKeyRoot*, char ∗ *buffer*)

Copy an ODB subtree in ASCII format from a buffer

Parameters:

- ***hDB*** ODB handle obtained via cm_get_experiment_database().
- ***hKeyRoot*** Handle for key where search starts, zero for root.
- ***buffer*** NULL-terminated buffer

Returns:

DB_SUCCESS, DB_TRUNCATED, DB_NO_MEMORY

Definition at line 5305 of file odb.c.

Referenced by db_create_record(), and db_load().

2.44.1.27 int db_paste_node (HNDLE *hDB*, HNDLE *hKeyRoot*, PMXML_NODE *node*)

Definition at line 5583 of file odb.c.

Referenced by db_paste_xml().

2.44.1.28 INT db_paste_xml (HNDLE *hDB*, HNDLE *hKeyRoot*, char ∗ *buffer*)

Paste an ODB subtree in XML format from a buffer

Parameters:

- ***hDB*** ODB handle obtained via cm_get_experiment_database().
- ***hKeyRoot*** Handle for key where search starts, zero for root.
- ***buffer*** NULL-terminated buffer

Returns:

DB_SUCCESS, DB_INVALID_PARAM, DB_NO_MEMORY, DB_TYPE_MISMATCH

Definition at line 5720 of file odb.c.

Referenced by db_load().

2.44.1.29 INT db_protect_database (HNDLE *hDB*)

Protect a database for read/write access outside of the **db_xxx** functions

Parameters:

- ***hDB*** ODB handle obtained via cm_get_experiment_database().

Returns:

DB_SUCCESS, DB_INVALID_HANDLE

Definition at line 1399 of file odb.c.

2.44.1.30 INT db_save (HNDLE *hDB*, HNDLE *hKey*, char ∗ *filename*, BOOL *bRemote*)

Save a branch of a database to an .ODB file

This function is used by the ODBEdit command save. For a description of the ASCII format, see db_copy(). Data of the whole ODB can be saved (hkey equal zero) or only a sub-tree.

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKey Handle for key where search starts, zero for root.

filename Filename of .ODB file.

bRemote Flag for saving database on remote server.

Returns:

DB_SUCCESS, DB_FILE_ERROR

Definition at line 5936 of file odb.c.

2.44.1.31 INT db_save_struct (HNDLE *hDB*, HNDLE *hKey*, char ∗ *file_name*, char ∗ *struct_name*, BOOL *append*)

Save a branch of a database to a C structure .H file

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKey Handle for key where search starts, zero for root.

file_name Filename of .ODB file.

struct_name Name of structure. If struct_name == NULL, the name of the key is used.

append If TRUE, append to end of existing file

Returns:

DB_SUCCESS, DB_INVALID_HANDLE, DB_FILE_ERROR

Definition at line 6191 of file odb.c.

2.44.1.32 INT db_save_xml (HNDLE *hDB*, HNDLE *hKey*, char ∗ *filename*)

Save a branch of a database to an .xml file

This function is used by the ODBEdit command save to write the contents of the ODB into a XML file. Data of the whole ODB can be saved (hkey equal zero) or only a sub-tree.

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKey Handle for key where search starts, zero for root.

filename Filename of .XML file.

Returns:

DB_SUCCESS, DB_FILE_ERROR

Definition at line 6138 of file odb.c.

2.44.1.33 INT db_save_xml_key (HNDLE *hDB*, HNDLE *hKey*, INT *level*, MXML_WRITER ∗ *writer*)

Definition at line 6035 of file odb.c.

Referenced by db_copy_xml(), and db_save_xml().

2.44.1.34 INT db_send_changed_records ()

Send all records to the ODB which were changed locally since the last call to this function.

This function is valid if used in conjunction with db_open_record() under the condition the record is open as MODE_WRITE access code.

- Full example dbchange.c which can be compiled as follow

```
gcc -DOS_LINUX -I/midas/include -o dbchange dbchange.c
/midas/linux/lib/libmidas.a -lutil}

\begin{verbatim}
//------- dbchange.c
#include "midas.h"
#include "msystem.h"
```

```
//-------- BOF dbchange.c
typedef struct {
    INT    my_number;
    float   my_rate;
} MY_STATISTICS;

MY_STATISTICS myrec;

#define MY_STATISTICS(_name) char *_name[] = {\
"My Number = INT : 0",\
"My Rate = FLOAT : 0",\
"",\
NULL }

HNDLE hDB, hKey;
```

```
// Main
int main(unsigned int argc,char **argv)
{
  char      host_name[HOST_NAME_LENGTH];
  char      expt_name[HOST_NAME_LENGTH];
  INT       lastnumber, status, msg;
  BOOL      debug=FALSE;
  char      i, ch;
  DWORD     update_time, mainlast_time;
  MY_STATISTICS (my_stat);

  // set default
  host_name[0] = 0;
  expt_name[0] = 0;

  // get default
  cm_get_environment(host_name, sizeof(host_name), expt_name, sizeof(expt_name));

  // get parameters
  for (i=1 ; i<argc ; i++)
  {
    if (argv[i][0] == ’-’ && argv[i][1] == ’d’)
      debug = TRUE;
    else if (argv[i][0] == ’-’)
    {
      if (i+1 >= argc || argv[i+1][0] == ’-’)
        goto usage;
      if (strncmp(argv[i],"-e",2) == 0)
        strcpy(expt_name, argv[++i]);
      else if (strncmp(argv[i],"-h",2)==0)
        strcpy(host_name, argv[++i]);
    }
    else
    {
   usage:
      printf("usage: dbchange [-h <Hostname>] [-e <Experiment>]\n");
      return 0;
    }
  }

  // connect to experiment
  status = cm_connect_experiment(host_name, expt_name, "dbchange", 0);
  if (status != CM_SUCCESS)
    return 1;

  // Connect to DB
  cm_get_experiment_database(&hDB, &hKey);

  // Create a default structure in ODB
  db_create_record(hDB, 0, "My statistics", strcomb(my_stat));

  // Retrieve key for that strucutre in ODB
  if (db_find_key(hDB, 0, "My statistics", &hKey) != DB_SUCCESS)
  {
    cm_msg(MERROR, "mychange", "cannot find My statistics");
    goto error;
  }

  // Hot link this structure in Write mode
  status = db_open_record(hDB, hKey, &myrec
                          , sizeof(MY_STATISTICS), MODE_WRITE, NULL, NULL);
  if (status != DB_SUCCESS)
  {
    cm_msg(MERROR, "mychange", "cannot open My statistics record");
    goto error;
  }

  // initialize ss_getchar()
  ss_getchar(0);

  // Main loop
  do
  {
    // Update local structure
    if ((ss_millitime() - update_time) > 100)
    {
      myrec.my_number += 1;
      if (myrec.my_number - lastnumber) {
        myrec.my_rate = 1000.f * (float) (myrec.my_number - lastnumber)
          / (float) (ss_millitime() - update_time);
      }
      update_time = ss_millitime();
      lastnumber = myrec.my_number;
    }

    // Publish local structure to ODB (db_send_changed_record)
    if ((ss_millitime() - mainlast_time) > 5000)
    {
      db_send_changed_records();                   // <------- Call
      mainlast_time = ss_millitime();
    }

    // Check for keyboard interaction
    ch = 0;
    while (ss_kbhit())
    {
      ch = ss_getchar(0);
      if (ch == -1)
        ch = getchar();
      if ((char) ch == ’!’)
        break;
    }
    msg = cm_yield(20);
  } while (msg != RPC_SHUTDOWN && msg != SS_ABORT && ch != ’!’);

 error:
  cm_disconnect_experiment();
  return 1;
}
//-------- EOF dbchange.c
```

Returns:

DB_SUCCESS

Definition at line 8251 of file odb.c.

Referenced by scan_fragment(), scheduler(), and tr_stop().

2.44.1.35 INT db_set_data (HNDLE *hDB*, HNDLE *hKey*, void ∗ *data*, INT *buf_size*, INT *num_values*, DWORD *type*)

Set key data from a handle. Adjust number of values if previous data has different size.

```
HNLDE hkey;
 INT   run_number;
 // get key handle for run number
 db_find_key(hDB, 0, "/Runinfo/Run number", &hkey);
 // set run number
 db_set_data(hDB, hkey, &run_number, sizeof(run_number),TID_INT);
```

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKey Handle for key where search starts, zero for root.

data Buffer from which data gets copied to.

buf_size Size of data buffer.

num_values Number of data values (for arrays).

type Type of key, one of TID_xxx (see Midas Data Types).

Returns:

DB_SUCCESS, DB_INVALID_HANDLE, DB_TRUNCATED

Definition at line 4330 of file odb.c.

Referenced by db_paste(), db_paste_node(), db_set_record(), and update_odb().

2.44.1.36 INT db_set_data_index (HNDLE *hDB*, HNDLE *hKey*, void ∗ *data*, INT *data_size*, INT *idx*, DWORD *type*)

Set key data for a key which contains an array of values.

This function sets individual values of a key containing an array. If the index is larger than the array size, the array is extended and the intermediate values are set to zero.

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKey Handle for key where search starts, zero for root.

data Pointer to single value of data.

data_size

idx Size of single data element.

type Type of key, one of TID_xxx (see Midas Data Types).

Returns:

DB_SUCCESS, DB_INVALID_HANDLE, DB_NO_ACCESS, DB_TYPE_MISMATCH

Definition at line 4554 of file odb.c.

Referenced by cm_register_transition(), db_paste_node(), and db_set_value_index().

2.44.1.37 INT db_set_record (HNDLE *hDB*, HNDLE *hKey*, void ∗ *data*, INT *buf_size*, INT *align*)

Copy a set of keys from local memory to the database.

An ODB sub-tree can be mapped to a C structure automatically via a hot-link using the function db_open_record() or manually with this function. Problems might occur if the ODB sub-tree contains values which don't match the C structure. Although the structure size is checked against the sub-tree size, no checking can be done if the type and order of the values in the structure are the same than those in the ODB sub-tree. Therefore it is recommended to use the function db_create_record() before using this function.

```
...
  memset(&lazyst,0,size);
  if (db_find_key(hDB, pLch->hKey, "Statistics",&hKeyst) == DB_SUCCESS)
    status = db_set_record(hDB, hKeyst, &lazyst, size, 0);
  else
    cm_msg(MERROR,"task","record %s/statistics not found", pLch->name)
...
```

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKey Handle for key where search starts, zero for root.

data Pointer where data is stored.

buf_size Size of data structure, must be obtained via sizeof(RECORD-NAME).

align Byte alignment calculated by the stub and passed to the rpc side to align data according to local machine. Must be zero when called from user level.

Returns:

DB_SUCCESS, DB_INVALID_HANDLE, DB_TYPE_MISMATCH, DB_STRUCT_SIZE_MISMATCH

Definition at line 6951 of file odb.c.

Referenced by al_check(), al_reset_alarm(), al_trigger_alarm(), db_open_record(), db_send_changed_records(), register_equipment(), and update_odb().

2.44.1.38 INT db_set_value (HNDLE *hDB*, HNDLE *hKeyRoot*, char ∗ *key_name*, void ∗ *data*, INT *data_size*, INT *num_values*, DWORD *type*)

Set value of a single key.

The function sets a single value or a whole array to a ODB key. Since the data buffer is of type void, no type checking can be performed by the compiler. Therefore the type has to be explicitly supplied, which is checked against the type stored in the ODB. key_name can contain the full path of a key (like: "/Equipment/Trigger/Settings/Level1") while hkey is zero which refers to the root, or hkey can refer to a sub-directory (like /Equipment/Trigger) and key_name is interpreted relative to that directory like "Settings/Level1".

```
INT level1;
  db_set_value(hDB, 0, "/Equipment/Trigger/Settings/Level1",
                          &level1, sizeof(level1), 1, TID_INT);
```

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKeyRoot Handle for key where search starts, zero for root.

key_name Name of key to search, can contain directories.

data Address of data.

data_size Size of data (in bytes).

num_values Number of data elements.

type Type of key, one of TID_xxx (see Midas Data Types)

Returns:

DB_SUCCESS, DB_INVALID_HANDLE, DB_NO_ACCESS, DB_TYPE_MISMATCH

Definition at line 2875 of file odb.c.

Referenced by al_trigger_alarm(), cm_connect_experiment1(), cm_delete_client_info(), cm_register_deferred_transition(), cm_register_transition(), cm_set_client_info(), cm_set_transition_sequence(), cm_set_watchdog_params(), cm_transition(), db_create_link(), db_get_value(), register_equipment(), tr_start(), and update_odb().

2.44.1.39 INT db_set_value_index (HNDLE *hDB*, HNDLE *hKeyRoot*, char ∗ *key_name*, void ∗ *data*, INT *data_size*, INT *index*, DWORD *type*, BOOL *truncate*)

Set single value of an array.

The function sets a single value of an ODB key which is an array. key_name can contain the full path of a key (like: "/Equipment/Trigger/Settings/Level1") while hkey is zero which refers to the root, or hkey can refer to a sub-directory (like

/Equipment/Trigger) and key_name is interpreted relative to that directory like "Settings/Level1".

```
INT level1;
  db_set_value_index(hDB, 0, "/Equipment/Trigger/Settings/Level1",
                          &level1, sizeof(level1), 15, TID_INT, FALSE);
```

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

hKeyRoot Handle for key where search starts, zero for root.

key_name Name of key to search, can contain directories.

data Address of data.

data_size Size of data (in bytes).

index Array index of value.

type Type of key, one of TID_xxx (see Midas Data Types)

truncate Truncate array to current index if TRUE

Returns:

<same as db_set_data_index>

Definition at line 3004 of file odb.c.

2.44.1.40 INT db_sprintf (char ∗ *string*, void ∗ *data*, INT *data_size*, INT *idx*, DWORD *type*)

Convert a database value to a string according to its type.

This function is a convenient way to convert a binary ODB value into a string depending on its type if is not known at compile time. If it is known, the normal sprintf() function can be used.

```
...
  for (j=0 ; j<key.num_values ; j++)
  {
    db_sprintf(pbuf, pdata, key.item_size, j, key.type);
    strcat(pbuf, "\n");
  }
  ...
```

Parameters:

string output ASCII string of data.

data Value data.

data_size Size of single data element.

idx Index for array data.

type Type of key, one of TID_xxx (see Midas Data Types).

Returns:
DB_SUCCESS

Definition at line 6365 of file odb.c.

Referenced by db_copy(), db_save_xml_key(), and hs_dump().

2.44.1.41 INT db_unlock_database (HNDLE *hDB*)

Unlock a database via system mutex calls.

Parameters:
hDB Handle to the database to unlock

Returns:
DB_SUCCESS, DB_INVALID_HANDLE

Definition at line 1370 of file odb.c.

Referenced by cm_check_client(), cm_cleanup(), cm_delete_client_info(), cm_get_watchdog_info(), cm_set_client_info(), cm_set_watchdog_params(), db_close_database(), db_create_key(), db_create_record(), db_delete_key1(), db_enum_key(), db_find_key(), db_get_data(), db_get_data_index(), db_get_key(), db_get_key_info(), db_get_key_time(), db_get_record(), db_get_record_size(), db_get_value(), db_open_database(), db_set_data(), db_set_data_index(), db_set_record(), and db_set_value().

2.44.1.42 INT db_update_record (INT *hDB*, INT *hKey*, int *s*)

If called locally, update a record (hDB/hKey) and copy its new contents to the local copy of it.

If called from a server, send a network notification to the client.

Parameters:
hDB ODB handle obtained via cm_get_experiment_database().

hKey Handle for key where search starts, zero for root.

s optional server socket

Returns:
DB_SUCCESS, DB_INVALID_HANDLE

Definition at line 8043 of file odb.c.

2.44.1.43 BOOL equal_ustring (char ∗ *str1*, char ∗ *str2*)

Definition at line 1428 of file odb.c.

Referenced by al_check(), bm_open_buffer(), cm_connect_client(), cm_connect_experiment1(), cm_exist(), cm_get_watchdog_info(), cm_list_experiments(), cm_set_client_info(), cm_shutdown(), db_check_record(), db_create_key(), db_find_key(), db_open_database(), db_paste(), db_paste_node(), logger_root(), and register_equipment().

2.44.1.44 char∗ extract_key (char ∗ *key_list*, char ∗ *key_name*)

Definition at line 1416 of file odb.c.

Referenced by db_create_key(), and db_find_key().

2.44.1.45 void xml_encode (char ∗ *src*, int *size*)

Definition at line 5992 of file odb.c.

2.45 Midas Alarm Functions (al_xxx)

Functions

- INT al_trigger_alarm (char ∗alarm_name, char ∗alarm_message, char ∗default_class, char ∗cond_str, INT type)
- INT al_reset_alarm (char ∗alarm_name)
- INT al_check ()

2.45.1 Function Documentation

2.45.1.1 INT al_check ()

Scan ODB for alarms.

Returns:

AL_SUCCESS

Definition at line 478 of file alarm.c.

Referenced by cm_yield().

2.45.1.2 INT al_reset_alarm (char ∗ *alarm_name*)

Reset (acknoledge) alarm.

Parameters:

alarm_name Alarm name, defined in /alarms/alarms

Returns:

AL_SUCCESS, AL_RESETE, AL_INVALID_NAME

Definition at line 390 of file alarm.c.

2.45.1.3 INT al_trigger_alarm (char ∗ *alarm_name*, char ∗ *alarm_message*, char ∗ *default_class*, char ∗ *cond_str*, INT *type*)

Trigger a certain alarm.

```
...
lazy.alarm[0] = 0;
size = sizeof(lazy.alarm);
db_get_value(hDB, pLch->hKey, "Settings/Alarm Class", lazy.alarm, &size, TID_STRING, TRUE);

// trigger alarm if defined
if (lazy.alarm[0])
  al_trigger_alarm("Tape", "Tape full...load new one!", lazy.alarm, "Tape full", AT_INTERNAL);
...
```

Parameters:

alarm_name Alarm name, defined in /alarms/alarms

alarm_message Optional message which goes with alarm

default_class If alarm is not yet defined under /alarms/alarms/<alarm_name>, a new one is created and this default class is used.

cond_str String displayed in alarm condition

type Alarm type, one of AT_xxx

Returns:

AL_SUCCESS, AL_INVALID_NAME

Definition at line 176 of file alarm.c.

Referenced by al_check().

2.46 Midas History Functions (hs_xxx)

Functions

- INT hs_set_path (char ∗path)

- INT hs_dump (DWORD event_id, DWORD start_time, DWORD end_time, DWORD interval, BOOL binary_time)

2.46.1 Function Documentation

2.46.1.1 INT hs_dump (DWORD *event_id*, DWORD *start_time*, DWORD *end_time*, DWORD *interval*, BOOL *binary_time*)

Display history for a given event at stdout. The output can be redirected to be read by Excel for example.

Parameters:

event_id Event ID

start_time Starting Date/Time

end_time End Date/Time

interval Minimum time in seconds between reported events. Can be used to skip events

binary_time Display DWORD time stamp

Returns:

HS_SUCCESS, HS_FILE_ERROR

Definition at line 1572 of file history.c.

2.46.1.2 INT hs_set_path (char ∗ *path*)

Sets the path for future history file accesses. Should be called before any other history function is called.

Parameters:

path Directory where history files reside

Returns:

HS_SUCCESS

Definition at line 49 of file history.c.

2.47 Midas Elog Functions (el_xxx)

Functions

- INT el_submit (int run, char ∗author, char ∗type, char ∗syst, char ∗subject, char ∗text, char ∗reply_to, char ∗encoding, char ∗afilename1, char ∗buffer1, INT buffer_size1, char ∗afilename2, char ∗buffer2, INT buffer_size2, char ∗afilename3, char ∗buffer3, INT buffer_size3, char ∗tag, INT tag_size)

2.47.1 Function Documentation

2.47.1.1 INT el_submit (int *run*, char ∗ *author*, char ∗ *type*, char ∗ *syst*, char ∗ *subject*, char ∗ *text*, char ∗ *reply_to*, char ∗ *encoding*, char ∗ *afilename1*, char ∗ *buffer1*, INT *buffer_size1*, char ∗ *afilename2*, char ∗ *buffer2*, INT *buffer_size2*, char ∗ *afilename3*, char ∗ *buffer3*, INT *buffer_size3*, char ∗ *tag*, INT *tag_size*)

Submit an ELog entry.

Parameters:

run Run Number.

author Message author.

type Message type.

syst Message system.

subject Subject.

text Message text.

reply_to In reply to this message.

encoding Text encoding, either HTML or plain.

afilename1 File name of attachment.

buffer1 File contents.

buffer_size1 Size of buffer in bytes.

afilename2 File name of attachment.

buffer2 File contents.

buffer_size2 Size of buffer in bytes.

afilename3 File name of attachment.

buffer3 File contents.

buffer_size3 Size of buffer in bytes.

tag If given, edit existing message.

tag_size Maximum size of tag.

Returns:
EL_SUCCESS

Definition at line 81 of file elog.c.

2.48 ∗ Camac Functions (Esone)

3 Midas Directory Documentation

3.1 /home/daqweb/midas/drivers/camac/ Directory Reference

Files

- file esone.c

3.2 /home/daqweb/midas/examples/custom/ Directory Reference

Files

- file myexpt.html
- file xcustom.odb

3.3 /home/daqweb/midas/drivers/ Directory Reference

Directories

- directorycamac

3.4 /home/daqweb/midas/examples/eventbuilder/ Directory Reference

Files

- file ebuser.c
- file mevb.c

3.5 /home/daqweb/midas/examples/ Directory Reference

Directories

- directorycustom
- directoryeventbuilder
- directoryexperiment

3.6 /home/daqweb/midas/examples/experiment/ Directory Reference

Files

- file adccalib.c
- file adcsum.c
- file analyzer.c
- file experim.h
- file frontend.c
- file scaler.c

3.7 /home/daqweb/midas/include/ Directory Reference

Files

- file mcstd.h
- file midas.h
- file mrpc.h

- file msystem.h
- file mvmestd.h
- file ybos.h

3.8 /home/daqweb/midas/src/ Directory Reference

Files

- file alarm.c
- file elog.c
- file history.c
- file mfe.c
- file midas.c
- file mrpc.c
- file odb.c
- file system.c
- file ybos.c

4 Midas Data Structure Documentation

4.1 ADC_CALIBRATION_PARAM Struct Reference

4.1.1 Field Documentation

4.1.1.1 double ADC_CALIBRATION_PARAM::histo_threshold

Definition at line 43 of file experim.h.

Referenced by adc_calib().

4.1.1.2 INT ADC_CALIBRATION_PARAM::pedestal[8]

Definition at line 41 of file experim.h.

Referenced by adc_calib().

4.1.1.3 float ADC_CALIBRATION_PARAM::software_gain[8]

Definition at line 42 of file experim.h.

Referenced by adc_calib().

4.2 ADC_SUMMING_PARAM Struct Reference

4.2.1 Field Documentation

4.2.1.1 float ADC_SUMMING_PARAM::adc_threshold

Definition at line 77 of file experim.h.

Referenced by adc_summing().

4.3 ALARM Struct Reference

4.3.1 Detailed Description

Alarm structure

Definition at line 1305 of file midas.h.

4.3.2 Field Documentation

4.3.2.1 BOOL ALARM::active

Definition at line 1306 of file midas.h.

Referenced by al_check(), and al_trigger_alarm().

4.3.2.2 char ALARM::alarm_class[32]

Definition at line 1314 of file midas.h.

Referenced by al_check(), al_reset_alarm(), and al_trigger_alarm().

4.3.2.3 char ALARM::alarm_message[80]

Definition at line 1315 of file midas.h.

Referenced by al_check(), and al_trigger_alarm().

4.3.2.4 INT ALARM::check_interval

Definition at line 1309 of file midas.h.

Referenced by al_check(), and al_trigger_alarm().

4.3.2.5 DWORD ALARM::checked_last

Definition at line 1310 of file midas.h.

Referenced by al_check(), al_reset_alarm(), and al_trigger_alarm().

4.3.2.6 char ALARM::condition[256]

Definition at line 1313 of file midas.h.

Referenced by al_check().

4.3.2.7 char ALARM::time_triggered_first[32]

Definition at line 1311 of file midas.h.

Referenced by al_reset_alarm(), and al_trigger_alarm().

4.3.2.8 char ALARM::time_triggered_last[32]

Definition at line 1312 of file midas.h.

Referenced by al_reset_alarm(), and al_trigger_alarm().

4.3.2.9 INT ALARM::triggered

Definition at line 1307 of file midas.h.

Referenced by al_reset_alarm(), and al_trigger_alarm().

4.3.2.10 INT ALARM::type

Definition at line 1308 of file midas.h.

Referenced by al_check(), and al_trigger_alarm().

4.4 ALARM_CLASS Struct Reference

4.4.1 Detailed Description

Alarm class structure

Definition at line 1275 of file midas.h.

4.4.2 Field Documentation

4.4.2.1 char ALARM_CLASS::display_bgcolor[32]

Definition at line 1284 of file midas.h.

4.4.2.2 char ALARM_CLASS::display_fgcolor[32]

Definition at line 1285 of file midas.h.

4.4.2.3 char ALARM_CLASS::execute_command[256]

Definition at line 1280 of file midas.h.

4.4.2.4 INT ALARM_CLASS::execute_interval

Definition at line 1281 of file midas.h.

4.4.2.5 DWORD ALARM_CLASS::execute_last

Definition at line 1282 of file midas.h.

Referenced by al_reset_alarm().

4.4.2.6 BOOL ALARM_CLASS::stop_run

Definition at line 1283 of file midas.h.

4.4.2.7 INT ALARM_CLASS::system_message_interval

Definition at line 1278 of file midas.h.

4.4.2.8 DWORD ALARM_CLASS::system_message_last

Definition at line 1279 of file midas.h.

Referenced by al_reset_alarm().

4.4.2.9 BOOL ALARM_CLASS::write_elog_message

Definition at line 1277 of file midas.h.

4.4.2.10 BOOL ALARM_CLASS::write_system_message

Definition at line 1276 of file midas.h.

4.5 ANA_MODULE Struct Reference

Data Fields

- char name [NAME_LENGTH]
- char author [NAME_LENGTH]
- INT(∗ analyzer)(EVENT_HEADER ∗, void ∗)
- INT(∗ bor)(INT run_number)
- INT(∗ eor)(INT run_number)
- INT(∗ init)()
- INT(∗ exit)()
- void ∗ parameters
- INT param_size
- char ∗∗ init_str
- BOOL enabled

4.5.1 Field Documentation

4.5.1.1 INT(∗ ANA_MODULE::analyzer)(EVENT_HEADER ∗, void ∗)

Pointer to user analyzer routine

4.5.1.2 char ANA_MODULE::author[NAME_LENGTH]

Author

Definition at line 1052 of file midas.h.

4.5.1.3 INT(∗ ANA_MODULE::bor)(INT run_number)

Pointer to begin-of-run routine

4.5.1.4 BOOL ANA_MODULE::enabled

Enabled flag

Definition at line 1062 of file midas.h.

4.5.1.5 INT(∗ ANA_MODULE::eor)(INT run_number)

Pointer to end-of-run routine

4.5.1.6 INT(∗ ANA_MODULE::exit)()

Pointer to exit routine

4.5.1.7 void∗ ANA_MODULE::histo_folder

Definition at line 1063 of file midas.h.

4.5.1.8 INT(∗ ANA_MODULE::init)()

Pointer to init routine

4.5.1.9 char∗∗ ANA_MODULE::init_str

Parameter init string

Definition at line 1061 of file midas.h.

4.5.1.10 char ANA_MODULE::name[NAME_LENGTH]

Module name

Definition at line 1051 of file midas.h.

4.5.1.11 INT ANA_MODULE::param_size

Size of parameter structure

Definition at line 1060 of file midas.h.

4.5.1.12 void∗ ANA_MODULE::parameters

Pointer to parameter structure

Definition at line 1059 of file midas.h.

4.6 ANA_OUTPUT_INFO Struct Reference

4.6.1 Field Documentation

4.6.1.1 BOOL ANA_OUTPUT_INFO::clear_histos

Definition at line 1114 of file midas.h.

4.6.1.2 BOOL ANA_OUTPUT_INFO::events_to_odb

Definition at line 1116 of file midas.h.

4.6.1.3 char ANA_OUTPUT_INFO::filename[256]

Definition at line 1110 of file midas.h.

4.6.1.4 char ANA_OUTPUT_INFO::global_memory_name[8]

Definition at line 1117 of file midas.h.

4.6.1.5 BOOL ANA_OUTPUT_INFO::histo_dump

Definition at line 1112 of file midas.h.

4.6.1.6 char ANA_OUTPUT_INFO::histo_dump_filename[256]

Definition at line 1113 of file midas.h.

4.6.1.7 char ANA_OUTPUT_INFO::last_histo_filename[256]

Definition at line 1115 of file midas.h.

4.6.1.8 BOOL ANA_OUTPUT_INFO::rwnt

Definition at line 1111 of file midas.h.

4.7 ANA_TEST Struct Reference

4.7.1 Field Documentation

4.7.1.1 DWORD ANA_TEST::count

Definition at line 1136 of file midas.h.

4.7.1.2 char ANA_TEST::name[80]

Definition at line 1134 of file midas.h.

4.7.1.3 DWORD ANA_TEST::previous_count

Definition at line 1137 of file midas.h.

4.7.1.4 BOOL ANA_TEST::registered

Definition at line 1135 of file midas.h.

4.7.1.5 BOOL ANA_TEST::value

Definition at line 1138 of file midas.h.

4.8 ANALYZE_REQUEST Struct Reference

Data Fields

- char event_name [NAME_LENGTH]
- AR_INFO ar_info
- INT(∗ analyzer)(EVENT_HEADER ∗, void ∗)
- ANA_MODULE ∗∗ ana_module
- BANK_LIST ∗ bank_list
- INT rwnt_buffer_size
- BOOL use_tests
- INT status
- HNDLE buffer_handle
- HNDLE request_id
- HNDLE hkey_variables
- HNDLE hkey_common
- void ∗ addr

- struct {
  } number
- DWORD events_received
- DWORD events_written

4.8.1 Field Documentation

4.8.1.1 void∗ ANALYZE_REQUEST::addr

Buffer for CWNT filling

Definition at line 1096 of file midas.h.

4.8.1.2 ANA_MODULE∗∗ ANALYZE_REQUEST::ana_module

List of analyzer modules

Definition at line 1086 of file midas.h.

4.8.1.3 INT(∗ ANALYZE_REQUEST::analyzer)(EVENT_HEADER ∗, void ∗)

Pointer to user analyzer routine

4.8.1.4 AR_INFO ANALYZE_REQUEST::ar_info

From above

Definition at line 1084 of file midas.h.

4.8.1.5 AR_STATS ANALYZE_REQUEST::ar_stats

Definition at line 1104 of file midas.h.

4.8.1.6 BANK_LIST∗ ANALYZE_REQUEST::bank_list

List of banks for event

Definition at line 1087 of file midas.h.

4.8.1.7 HNDLE ANALYZE_REQUEST::buffer_handle

MIDAS buffer handle

Definition at line 1092 of file midas.h.

4.8.1.8 char ANALYZE_REQUEST::event_name[NAME_LENGTH]

Event name

Definition at line 1083 of file midas.h.

4.8.1.9 DWORD ANALYZE_REQUEST::events_received

number of events sent

Definition at line 1102 of file midas.h.

4.8.1.10 DWORD ANALYZE_REQUEST::events_written

number of events written

Definition at line 1103 of file midas.h.

4.8.1.11 HNDLE ANALYZE_REQUEST::hkey_common

Key to common subtree

Definition at line 1095 of file midas.h.

4.8.1.12 HNDLE ANALYZE_REQUEST::hkey_variables

Key to variables subtree in ODB

Definition at line 1094 of file midas.h.

4.8.1.13 char** ANALYZE_REQUEST::init_string

Definition at line 1090 of file midas.h.

4.8.1.14 struct { ... } ANALYZE_REQUEST::number

Buffer for event number for CWNT

4.8.1.15 HNDLE ANALYZE_REQUEST::request_id

Event request handle

Definition at line 1093 of file midas.h.

4.8.1.16 DWORD ANALYZE_REQUEST::run

Definition at line 1098 of file midas.h.

4.8.1.17 INT ANALYZE_REQUEST::rwnt_buffer_size

Size in events of RW N-tuple buf

Definition at line 1088 of file midas.h.

4.8.1.18 DWORD ANALYZE_REQUEST::serial

Definition at line 1099 of file midas.h.

4.8.1.19 INT ANALYZE_REQUEST::status

One of FE_xxx

Definition at line 1091 of file midas.h.

4.8.1.20 DWORD ANALYZE_REQUEST::time

Definition at line 1100 of file midas.h.

4.8.1.21 BOOL ANALYZE_REQUEST::use_tests

Use tests for this event

Definition at line 1089 of file midas.h.

4.9 AR_INFO Struct Reference

Data Fields

- INT event_id
- INT trigger_mask
- INT sampling_type
- char buffer [NAME_LENGTH]
- BOOL enabled
- char client_name [NAME_LENGTH]
- char host [NAME_LENGTH]

4.9.1 Field Documentation

4.9.1.1 char AR_INFO::buffer[NAME_LENGTH]

Event buffer to send events into

Definition at line 1070 of file midas.h.

4.9.1.2 char AR_INFO::client_name[NAME_LENGTH]

Analyzer name

Definition at line 1072 of file midas.h.

4.9.1.3 BOOL AR_INFO::enabled

Enable flag

Definition at line 1071 of file midas.h.

4.9.1.4 INT AR_INFO::event_id

Event ID associated with equipm.

Definition at line 1067 of file midas.h.

4.9.1.5 char AR_INFO::host[NAME_LENGTH]

Host on which analyzer is running

Definition at line 1073 of file midas.h.

4.9.1.6 INT AR_INFO::sampling_type

GET_ALL/GET_SOME

Definition at line 1069 of file midas.h.

4.9.1.7 INT AR_INFO::trigger_mask

Trigger mask

Definition at line 1068 of file midas.h.

4.10 AR_STATS Struct Reference

4.10.1 Field Documentation

4.10.1.1 double AR_STATS::events_per_sec

Definition at line 1078 of file midas.h.

4.10.1.2 double AR_STATS::events_received

Definition at line 1077 of file midas.h.

4.10.1.3 double AR_STATS::events_written

Definition at line 1079 of file midas.h.

4.11 ASUM_BANK Struct Reference

4.11.1 Field Documentation

4.11.1.1 float ASUM_BANK::average

Definition at line 110 of file experim.h.

Referenced by adc_summing().

4.11.1.2 float ASUM_BANK::sum

Definition at line 109 of file experim.h.

Referenced by adc_summing().

4.12 BANK Struct Reference

Data Fields

- char name [4]
- WORD type
- WORD data_size

4.12.1 Field Documentation

4.12.1.1 WORD BANK::data_size

-

Definition at line 1017 of file midas.h.

Referenced by bk_close(), bk_create(), bk_find(), bk_iterate(), bk_locate(), and bk_swap().

4.12.1.2 char BANK::name[4]

-

Definition at line 1015 of file midas.h.

Referenced by bk_close(), bk_create(), bk_find(), bk_list(), bk_locate(), and update_odb().

4.12.1.3 WORD BANK::type

-

Definition at line 1016 of file midas.h.

Referenced by bk_close(), bk_create(), bk_find(), bk_locate(), bk_swap(), and update_odb().

4.13 BANK32 Struct Reference

Data Fields

- char name [4]
- DWORD type
- DWORD data_size

4.13.1 Field Documentation

4.13.1.1 DWORD BANK32::data_size

-

Definition at line 1023 of file midas.h.

Referenced by bk_close(), bk_create(), bk_find(), bk_locate(), and bk_swap().

4.13.1.2 char BANK32::name[4]

-

Definition at line 1021 of file midas.h.

Referenced by bk_close(), bk_create(), bk_find(), bk_list(), bk_locate(), and update_odb().

4.13.1.3 DWORD BANK32::type

-

Definition at line 1022 of file midas.h.

Referenced by bk_close(), bk_create(), bk_find(), bk_locate(), bk_swap(), and update_odb().

4.14 BANK_HEADER Struct Reference

Data Fields

- DWORD data_size
- DWORD flags

4.14.1 Field Documentation

4.14.1.1 DWORD BANK_HEADER::data_size

Size in bytes

Definition at line 1010 of file midas.h.

Referenced by bk_find(), bk_swap(), eb_mfragment_add(), eb_yfragment_add(), and source_scan().

4.14.1.2 DWORD BANK_HEADER::flags

internal flag

Definition at line 1011 of file midas.h.

Referenced by bk_swap().

4.15 BANK_LIST Struct Reference

Data Fields

- char name [9]
- WORD type
- DWORD size
- char ∗∗ init_str
- BOOL output_flag
- void ∗ addr
- DWORD n_data
- HNDLE def_key

4.15.1 Field Documentation

4.15.1.1 void∗ BANK_LIST::addr

-

Definition at line 1038 of file midas.h.

4.15.1.2 HNDLE BANK_LIST::def_key

-

Definition at line 1040 of file midas.h.

4.15.1.3 char∗∗ BANK_LIST::init_str

-

Definition at line 1036 of file midas.h.

Referenced by register_equipment().

4.15.1.4 DWORD BANK_LIST::n_data

-

Definition at line 1039 of file midas.h.

4.15.1.5 char BANK_LIST::name[9]

-

Definition at line 1033 of file midas.h.

Referenced by register_equipment().

4.15.1.6 BOOL BANK_LIST::output_flag

-

Definition at line 1037 of file midas.h.

4.15.1.7 DWORD BANK_LIST::size

-

Definition at line 1035 of file midas.h.

4.15.1.8 WORD BANK_LIST::type

-

Definition at line 1034 of file midas.h.

Referenced by register_equipment().

4.16 BUFFER Struct Reference

Data Fields

- BOOL attached
- INT client_index
- BUFFER_HEADER $*$ buffer_header

- void ∗ buffer_data
- char ∗ read_cache
- INT read_cache_size
- INT read_cache_rp
- INT read_cache_wp
- char ∗ write_cache
- INT write_cache_size
- INT write_cache_rp
- INT write_cache_wp
- HNDLE mutex
- INT shm_handle
- INT index
- BOOL callback

4.16.1 Field Documentation

4.16.1.1 BOOL BUFFER::attached

TRUE if buffer is attached

Definition at line 860 of file midas.h.

Referenced by bm_check_buffers(), bm_close_buffer(), bm_empty_buffers(), bm_flush_cache(), bm_open_buffer(), bm_push_event(), bm_receive_event(), bm_remove_event_request(), bm_send_event(), bm_set_cache_size(), bm_skip_event(), cm_cleanup(), and cm_set_watchdog_params().

4.16.1.2 void∗ BUFFER::buffer_data

pointer to buffer data

Definition at line 863 of file midas.h.

Referenced by bm_open_buffer().

4.16.1.3 BUFFER_HEADER∗ BUFFER::buffer_header

pointer to buffer header

Definition at line 862 of file midas.h.

Referenced by bm_check_buffers(), bm_close_buffer(), bm_empty_buffers(), bm_flush_cache(), bm_open_buffer(), bm_push_event(), bm_receive_event(), bm_remove_event_request(), bm_send_event(), bm_skip_event(), bm_validate_client_index(), bm_wait_for_free_space(), cm_cleanup(), and cm_set_watchdog_params().

4.16.1.4 BOOL BUFFER::callback

callback defined for this buffer

Definition at line 875 of file midas.h.

Referenced by bm_open_buffer(), and bm_push_event().

4.16.1.5 INT BUFFER::client_index

index to CLIENT str. in buf.

Definition at line 861 of file midas.h.

Referenced by bm_open_buffer(), bm_validate_client_index(), and cm_cleanup().

4.16.1.6 INT BUFFER::index

connection index / tid

Definition at line 874 of file midas.h.

Referenced by bm_check_buffers(), bm_close_buffer(), bm_empty_buffers(), bm_open_buffer(), and cm_set_watchdog_params().

4.16.1.7 HNDLE BUFFER::mutex

mutex/semaphore handle

Definition at line 872 of file midas.h.

4.16.1.8 char* BUFFER::read_cache

cache for burst read

Definition at line 864 of file midas.h.

Referenced by bm_copy_from_cache(), bm_dispatch_from_cache(), bm_push_event(), bm_receive_event(), and bm_set_cache_size().

4.16.1.9 INT BUFFER::read_cache_rp

cache read pointer

Definition at line 866 of file midas.h.

Referenced by bm_copy_from_cache(), bm_dispatch_from_cache(), bm_empty_buffers(), bm_read_cache_has_events(), bm_set_cache_size(), and bm_skip_event().

4.16.1.10 INT BUFFER::read_cache_size

cache size in bytes

Definition at line 865 of file midas.h.

Referenced by bm_close_buffer(), bm_push_event(), bm_read_cache_has_events(), bm_receive_event(), and bm_set_cache_size().

4.16.1.11 INT BUFFER::read_cache_wp

cache write pointer

Definition at line 867 of file midas.h.

Referenced by bm_copy_from_cache(), bm_dispatch_from_cache(), bm_empty_buffers(), bm_push_event(), bm_read_cache_has_events(), bm_receive_event(), bm_set_cache_size(), and bm_skip_event().

4.16.1.12 INT BUFFER::shm_handle

handle to shared memory

Definition at line 873 of file midas.h.

Referenced by bm_close_buffer(), and bm_open_buffer().

4.16.1.13 char* BUFFER::write_cache

cache for burst read

Definition at line 868 of file midas.h.

Referenced by bm_flush_cache(), bm_send_event(), and bm_set_cache_size().

4.16.1.14 INT BUFFER::write_cache_rp

cache read pointer

Definition at line 870 of file midas.h.

Referenced by bm_flush_cache(), and bm_set_cache_size().

4.16.1.15 INT BUFFER::write_cache_size

cache size in bytes

Definition at line 869 of file midas.h.

Referenced by bm_close_buffer(), bm_flush_cache(), bm_send_event(), and bm_set_cache_size().

4.16.1.16 INT BUFFER::write_cache_wp

cache write pointer

Definition at line 871 of file midas.h.

Referenced by bm_flush_cache(), bm_send_event(), and bm_set_cache_size().

4.17 BUFFER_CLIENT Struct Reference

Data Fields

- char name [NAME_LENGTH]
- INT pid
- INT tid
- INT thandle
- INT port
- INT read_pointer
- INT max_request_index
- INT num_received_events
- INT num_sent_events
- INT num_waiting_events
- float data_rate
- BOOL read_wait
- INT write_wait
- BOOL wake_up
- BOOL all_flag
- DWORD last_activity
- DWORD watchdog_timeout

4.17.1 Field Documentation

4.17.1.1 BOOL BUFFER_CLIENT::all_flag

at least one GET_ALL request

Definition at line 835 of file midas.h.

Referenced by bm_remove_event_request().

4.17.1.2 float BUFFER_CLIENT::data_rate

data rate in kB/sec

Definition at line 831 of file midas.h.

4.17.1.3 EVENT_REQUEST BUFFER_CLIENT::event_request[MAX_EVENT_REQUESTS]

Definition at line 839 of file midas.h.

Referenced by bm_push_event(), bm_receive_event(), bm_remove_event_request(), bm_send_event(), and bm_wait_for_free_space().

4.17.1.4 DWORD BUFFER_CLIENT::last_activity

time of last activity

Definition at line 836 of file midas.h.

Referenced by bm_open_buffer(), cm_cleanup(), and cm_set_watchdog_params().

4.17.1.5 INT BUFFER_CLIENT::max_request_index

index of last request

Definition at line 827 of file midas.h.

Referenced by bm_push_event(), bm_receive_event(), bm_remove_event_request(), bm_send_event(), and bm_wait_for_free_space().

4.17.1.6 char BUFFER_CLIENT::name[NAME_LENGTH]

name of client

Definition at line 821 of file midas.h.

Referenced by bm_open_buffer(), bm_validate_client_index(), bm_validate_client_pointers(), and cm_cleanup().

4.17.1.7 INT BUFFER_CLIENT::num_received_events

no of received events

Definition at line 828 of file midas.h.

4.17.1.8 INT BUFFER_CLIENT::num_sent_events

no of sent events

Definition at line 829 of file midas.h.

4.17.1.9 INT BUFFER_CLIENT::num_waiting_events

no of waiting events

Definition at line 830 of file midas.h.

Referenced by bm_send_event().

4.17.1.10 INT BUFFER_CLIENT::pid

process ID

Definition at line 822 of file midas.h.

Referenced by bm_close_buffer(), bm_flush_cache(), bm_open_buffer(), bm_send_event(), bm_update_read_pointer(), bm_validate_client_index(), bm_wait_for_free_space(), bm_wakeup_producers(), and cm_cleanup().

4.17.1.11 INT BUFFER_CLIENT::port

UDP port for wake up

Definition at line 825 of file midas.h.

Referenced by bm_close_buffer(), bm_open_buffer(), bm_wait_for_free_space(), bm_wakeup_producers(), and cm_cleanup().

4.17.1.12 INT BUFFER_CLIENT::read_pointer

read pointer to buffer

Definition at line 826 of file midas.h.

Referenced by bm_empty_buffers(), bm_flush_cache(), bm_open_buffer(), bm_push_event(), bm_receive_event(), bm_send_event(), bm_skip_event(), bm_update_read_pointer(), bm_validate_client_pointers(), bm_wait_for_free_space(), and bm_wakeup_producers().

4.17.1.13 BOOL BUFFER_CLIENT::read_wait

wait for read - flag

Definition at line 832 of file midas.h.

Referenced by bm_close_buffer(), bm_flush_cache(), bm_receive_event(), bm_send_event(), bm_wait_for_free_space(), and cm_cleanup().

4.17.1.14 INT BUFFER_CLIENT::thandle

thread handle

Definition at line 824 of file midas.h.

Referenced by bm_open_buffer().

4.17.1.15 INT BUFFER_CLIENT::tid

thread ID

Definition at line 823 of file midas.h.

Referenced by bm_open_buffer(), and bm_wakeup_producers().

4.17.1.16 BOOL BUFFER_CLIENT::wake_up

client got a wake-up msg

Definition at line 834 of file midas.h.

4.17.1.17 DWORD BUFFER_CLIENT::watchdog_timeout

timeout in ms

Definition at line 837 of file midas.h.

Referenced by bm_open_buffer(), cm_cleanup(), and cm_set_watchdog_params().

4.17.1.18 INT BUFFER_CLIENT::write_wait

wait for write # bytes

Definition at line 833 of file midas.h.

Referenced by bm_close_buffer(), bm_wait_for_free_space(), bm_wakeup_producers(), and cm_cleanup().

4.18 BUFFER_HEADER Struct Reference

Data Fields

- char name [NAME_LENGTH]
- INT num_clients
- INT max_client_index
- INT size
- INT read_pointer
- INT write_pointer
- INT num_in_events
- INT num_out_events
- BUFFER_CLIENT client [MAX_CLIENTS]

4.18.1 Field Documentation

4.18.1.1 BUFFER_CLIENT BUFFER_HEADER::client[MAX_CLIENTS]

entries for clients

Definition at line 853 of file midas.h.

Referenced by bm_close_buffer(), bm_flush_cache(), bm_open_buffer(), bm_push_event(), bm_receive_event(), bm_send_event(), bm_skip_event(), bm_update_read_pointer(), bm_validate_client_index(), bm_wait_for_free_space(), bm_wakeup_producers(), cm_cleanup(), and cm_set_watchdog_params().

4.18.1.2 INT BUFFER_HEADER::max_client_index

index of last client

Definition at line 846 of file midas.h.

Referenced by bm_close_buffer(), bm_flush_cache(), bm_open_buffer(), bm_send_event(), bm_update_read_pointer(), bm_validate_client_index(), bm_wait_for_free_space(), bm_wakeup_producers(), and cm_cleanup().

4.18.1.3 char BUFFER_HEADER::name[NAME_LENGTH]

name of buffer

Definition at line 844 of file midas.h.

Referenced by bm_check_buffers(), bm_close_buffer(), bm_flush_cache(), bm_open_buffer(), bm_push_event(), bm_send_event(), bm_validate_client_index(), bm_validate_client_pointers(), bm_wait_for_free_space(), and cm_cleanup().

4.18.1.4 INT BUFFER_HEADER::num_clients

no of active clients

Definition at line 845 of file midas.h.

Referenced by bm_close_buffer(), bm_open_buffer(), and cm_cleanup().

4.18.1.5 INT BUFFER_HEADER::num_in_events

no of received events

Definition at line 850 of file midas.h.

Referenced by bm_flush_cache(), and bm_send_event().

4.18.1.6 INT BUFFER_HEADER::num_out_events

no of distributed events

Definition at line 851 of file midas.h.

Referenced by bm_push_event(), and bm_receive_event().

4.18.1.7 INT BUFFER_HEADER::read_pointer

read pointer

Definition at line 848 of file midas.h.

Referenced by bm_flush_cache(), bm_receive_event(), bm_send_event(), bm_update_read_pointer(), bm_validate_client_pointers(), bm_wait_for_free_space(), and bm_wakeup_producers().

4.18.1.8 INT BUFFER_HEADER::size

size of data area in bytes

Definition at line 847 of file midas.h.

Referenced by bm_flush_cache(), bm_open_buffer(), bm_push_event(), bm_receive_event(), bm_send_event(), bm_update_read_pointer(), bm_validate_client_pointers(), bm_wait_for_free_space(), and bm_wakeup_producers().

4.18.1.9 INT BUFFER_HEADER::write_pointer

write pointer

Definition at line 849 of file midas.h.

Referenced by bm_flush_cache(), bm_open_buffer(), bm_push_event(), bm_receive_event(), bm_send_event(), bm_skip_event(), bm_update_read_pointer(), bm_validate_client_pointers(), bm_wait_for_free_space(), and bm_wakeup_producers().

4.19 BUS_DRIVER Struct Reference

Data Fields

- char name [NAME_LENGTH]
- INT(∗ bd)(INT cmd,...)
- void ∗ bd_info

4.19.1 Field Documentation

4.19.1.1 INT(∗ BUS_DRIVER::bd)(INT cmd,...)

Device driver entry point

4.19.1.2 void∗ BUS_DRIVER::bd_info

Private info for bus driver

Definition at line 918 of file midas.h.

4.19.1.3 char BUS_DRIVER::name[NAME_LENGTH]

Driver name

Definition at line 916 of file midas.h.

4.20 DATABASE Struct Reference

4.20.1 Field Documentation

4.20.1.1 BOOL DATABASE::attached

Definition at line 375 of file msystem.h.

Referenced by cm_check_client(), cm_cleanup(), cm_get_watchdog_info(), cm_set_watchdog_params(), db_close_database(), db_create_key(), db_delete_key1(), db_enum_key(), db_find_key(), db_get_data(), db_get_data_index(), db_get_key(), db_get_key_info(), db_get_key_time(), db_get_value(), db_open_database(), db_set_data(), and db_set_data_index().

4.20.1.2 INT DATABASE::client_index

Definition at line 376 of file msystem.h.

Referenced by cm_cleanup(), cm_set_watchdog_params(), db_close_database(), and db_open_database().

4.20.1.3 void∗ DATABASE::database_data

Definition at line 378 of file msystem.h.

Referenced by db_open_database().

4.20.1.4 DATABASE_HEADER∗ DATABASE::database_header

Definition at line 377 of file msystem.h.

Referenced by cm_check_client(), cm_cleanup(), cm_get_watchdog_info(), cm_set_watchdog_params(), db_close_database(), db_create_key(), db_delete_key1(), db_enum_key(), db_find_key(), db_get_data(), db_get_data_index(), db_get_key(), db_get_key_info(), db_get_key_time(), db_get_value(), db_lock_database(), db_open_database(), db_protect_database(), db_set_data(), db_set_data_index(), db_set_value(), and db_unlock_database().

4.20.1.5 INT DATABASE::index

Definition at line 382 of file msystem.h.

Referenced by cm_set_watchdog_params(), db_close_database(), and db_open_database().

4.20.1.6 INT DATABASE::lock_cnt

Definition at line 380 of file msystem.h.

Referenced by db_lock_database(), db_open_database(), and db_unlock_database().

4.20.1.7 HNDLE DATABASE::mutex

Definition at line 379 of file msystem.h.

4.20.1.8 char DATABASE::name[NAME_LENGTH]

Definition at line 374 of file msystem.h.

4.20.1.9 BOOL DATABASE::protect

Definition at line 383 of file msystem.h.

Referenced by db_lock_database(), db_open_database(), db_protect_database(), and db_unlock_database().

4.20.1.10 HNDLE DATABASE::shm_handle

Definition at line 381 of file msystem.h.

Referenced by db_close_database(), and db_open_database().

4.21 DATABASE_CLIENT Struct Reference

4.21.1 Field Documentation

4.21.1.1 DWORD DATABASE_CLIENT::last_activity

Definition at line 348 of file msystem.h.

Referenced by cm_cleanup(), cm_get_watchdog_info(), cm_set_watchdog_params(), and db_open_database().

4.21.1.2 INT DATABASE_CLIENT::max_index

Definition at line 350 of file msystem.h.

Referenced by db_close_database(), and db_open_database().

4.21.1.3 char DATABASE_CLIENT::name[NAME_LENGTH]

Definition at line 342 of file msystem.h.

Referenced by cm_cleanup(), cm_get_watchdog_info(), cm_transition(), and db_open_database().

4.21.1.4 INT DATABASE_CLIENT::num_open_records

Definition at line 347 of file msystem.h.

Referenced by db_open_database().

4.21.1.5 OPEN_RECORD DATABASE_CLIENT::open_record[MAX_OPEN_RECORDS]

Definition at line 352 of file msystem.h.

Referenced by cm_cleanup(), db_close_database(), and db_open_database().

4.21.1.6 INT DATABASE_CLIENT::pid

Definition at line 343 of file msystem.h.

Referenced by cm_cleanup(), cm_get_watchdog_info(), db_close_database(), and db_open_database().

4.21.1.7 INT DATABASE_CLIENT::port

Definition at line 346 of file msystem.h.

Referenced by db_open_database().

4.21.1.8 INT DATABASE_CLIENT::thandle

Definition at line 345 of file msystem.h.

Referenced by db_open_database().

4.21.1.9 INT DATABASE_CLIENT::tid

Definition at line 344 of file msystem.h.

Referenced by cm_check_client(), cm_cleanup(), and db_open_database().

4.21.1.10 DWORD DATABASE_CLIENT::watchdog_timeout

Definition at line 349 of file msystem.h.

Referenced by cm_cleanup(), cm_get_watchdog_info(), cm_set_watchdog_params(), and db_open_database().

4.22 DATABASE_HEADER Struct Reference

4.22.1 Field Documentation

4.22.1.1 DATABASE_CLIENT DATABASE_HEADER::client[MAX_CLIENTS]

Definition at line 367 of file msystem.h.

Referenced by cm_check_client(), cm_cleanup(), cm_get_watchdog_info(), cm_set_watchdog_params(), db_close_database(), and db_open_database().

4.22.1.2 INT DATABASE_HEADER::data_size

Definition at line 362 of file msystem.h.

Referenced by db_close_database(), and db_open_database().

4.22.1.3 INT DATABASE_HEADER::first_free_data

Definition at line 365 of file msystem.h.

Referenced by db_open_database().

4.22.1.4 INT DATABASE_HEADER::first_free_key

Definition at line 364 of file msystem.h.

Referenced by db_open_database().

4.22.1.5 INT DATABASE_HEADER::key_size

Definition at line 361 of file msystem.h.

Referenced by db_open_database().

4.22.1.6 INT DATABASE_HEADER::max_client_index

Definition at line 360 of file msystem.h.

Referenced by cm_check_client(), cm_cleanup(), cm_get_watchdog_info(), db_close_database(), and db_open_database().

4.22.1.7 char DATABASE_HEADER::name[NAME_LENGTH]

Definition at line 357 of file msystem.h.

Referenced by cm_cleanup(), db_close_database(), and db_open_database().

4.22.1.8 INT DATABASE_HEADER::num_clients

Definition at line 359 of file msystem.h.

Referenced by cm_cleanup(), db_close_database(), and db_open_database().

4.22.1.9 INT DATABASE_HEADER::root_key

Definition at line 363 of file msystem.h.

Referenced by db_create_key(), db_delete_key1(), db_enum_key(), db_find_key(), db_get_key(), and db_open_database().

4.22.1.10 INT DATABASE_HEADER::version

Definition at line 358 of file msystem.h.

Referenced by db_open_database().

4.23 DD_MT_BUFFER Struct Reference

Data Fields

- INT n_channels
- midas_thread_t thread_id
- INT status
- DD_MT_CHANNEL ∗ channel

4.23.1 Field Documentation

4.23.1.1 DD_MT_CHANNEL∗ DD_MT_BUFFER::channel

One data set for each channel

Definition at line 930 of file midas.h.

Referenced by device_driver(), and sc_thread().

4.23.1.2 INT DD_MT_BUFFER::n_channels

Number of channels

Definition at line 927 of file midas.h.

Referenced by device_driver().

4.23.1.3 INT DD_MT_BUFFER::status

Status passed from device thread

Definition at line 929 of file midas.h.

Referenced by device_driver(), and sc_thread().

4.23.1.4 midas_thread_t DD_MT_BUFFER::thread_id

Thread ID

Definition at line 928 of file midas.h.

Referenced by device_driver().

4.24 DD_MT_CHANNEL Struct Reference

Data Fields

- float array [CMD_GET_LAST]
- char label [NAME_LENGTH]

4.24.1 Field Documentation

4.24.1.1 float DD_MT_CHANNEL::array[CMD_GET_LAST]

Array for various values

Definition at line 922 of file midas.h.

Referenced by device_driver(), and sc_thread().

4.24.1.2 char DD_MT_CHANNEL::label[NAME_LENGTH]

Array for channel labels

Definition at line 923 of file midas.h.

Referenced by device_driver().

4.25 DEF_RECORD Struct Reference

4.25.1 Field Documentation

4.25.1.1 DWORD DEF_RECORD::def_offset

Definition at line 1172 of file midas.h.

4.25.1.2 DWORD DEF_RECORD::event_id

Definition at line 1170 of file midas.h.

4.25.1.3 char DEF_RECORD::event_name[NAME_LENGTH]

Definition at line 1171 of file midas.h.

4.26 DEVICE_DRIVER Struct Reference

Data Fields

- char name [NAME_LENGTH]
- INT(∗ dd)(INT cmd,...)
- INT channels
- INT(∗ bd)(INT cmd,...)
- DWORD flags
- void ∗ dd_info
- DD_MT_BUFFER ∗ mt_buffer
- INT stop_thread
- HNDLE mutex

4.26.1 Field Documentation

4.26.1.1 INT(∗ DEVICE_DRIVER::bd)(INT cmd,...)

Bus driver entry point

Referenced by device_driver().

4.26.1.2 INT DEVICE_DRIVER::channels

Number of channels

Definition at line 937 of file midas.h.

Referenced by device_driver(), and sc_thread().

4.26.1.3 INT(∗ DEVICE_DRIVER::dd)(INT cmd,...)

Device driver entry point

Referenced by device_driver(), and sc_thread().

4.26.1.4 void∗ DEVICE_DRIVER::dd_info

Private info for device driver

Definition at line 940 of file midas.h.

Referenced by device_driver(), and sc_thread().

4.26.1.5 DWORD DEVICE_DRIVER::flags

Combination of DF_xx

Definition at line 939 of file midas.h.

Referenced by device_driver(), and main().

4.26.1.6 DD_MT_BUFFER∗ DEVICE_DRIVER::mt_buffer

pointer to multithread buffer

Definition at line 941 of file midas.h.

Referenced by device_driver(), and sc_thread().

4.26.1.7 HNDLE DEVICE_DRIVER::mutex

mutex/semaphore handle for buffer

Definition at line 943 of file midas.h.

Referenced by device_driver(), and sc_thread().

4.26.1.8 char DEVICE_DRIVER::name[NAME_LENGTH]

Driver name

Definition at line 935 of file midas.h.

Referenced by device_driver(), main(), and register_equipment().

4.26.1.9 INT DEVICE_DRIVER::stop_thread

flag used to stop the thread

Definition at line 942 of file midas.h.

Referenced by device_driver(), and sc_thread().

4.27 eqpmnt Struct Reference

Data Fields

- char name [NAME_LENGTH]
- EQUIPMENT_INFO info
- INT(∗ readout)(char ∗, INT)
- INT(∗ cd)(INT cmd, PEQUIPMENT)
- DEVICE_DRIVER ∗ driver
- void ∗ event_descrip

- void ∗ cd_info
- INT status
- DWORD last_called
- DWORD last_idle
- DWORD poll_count
- INT format
- HNDLE buffer_handle
- HNDLE hkey_variables
- DWORD serial_number
- DWORD subevent_number
- DWORD odb_out
- DWORD odb_in
- DWORD bytes_sent
- DWORD events_sent

4.27.1 Field Documentation

4.27.1.1 HNDLE eqpmnt::buffer_handle

MIDAS buffer handle

Definition at line 987 of file midas.h.

Referenced by receive_trigger_event(), register_equipment(), scheduler(), send_event(), and tr_stop().

4.27.1.2 DWORD eqpmnt::bytes_sent

number of bytes sent

Definition at line 993 of file midas.h.

Referenced by receive_trigger_event(), scan_fragment(), scheduler(), send_event(), source_scan(), and tr_stop().

4.27.1.3 INT(∗ eqpmnt::cd)(INT cmd, PEQUIPMENT)

Class driver routine

Referenced by main(), register_equipment(), and scheduler().

4.27.1.4 void∗ eqpmnt::cd_info

private data for class driver

Definition at line 981 of file midas.h.

4.27.1.5 DEVICE_DRIVER* eqpmnt::driver

Device driver list

Definition at line 979 of file midas.h.

Referenced by main(), and register_equipment().

4.27.1.6 void* eqpmnt::event_descrip

Init string for fixed events or bank list

Definition at line 980 of file midas.h.

Referenced by register_equipment().

4.27.1.7 DWORD eqpmnt::events_sent

number of events sent

Definition at line 994 of file midas.h.

Referenced by display(), receive_trigger_event(), scan_fragment(), scheduler(), send_event(), source_scan(), and tr_stop().

4.27.1.8 INT eqpmnt::format

FORMAT_xxx

Definition at line 986 of file midas.h.

Referenced by load_fragment(), receive_trigger_event(), register_equipment(), and scheduler().

4.27.1.9 HNDLE eqpmnt::hkey_variables

Key to variables subtree in ODB

Definition at line 988 of file midas.h.

Referenced by receive_trigger_event(), register_equipment(), and scheduler().

4.27.1.10 EQUIPMENT_INFO eqpmnt::info

From above

Definition at line 976 of file midas.h.

Referenced by display(), interrupt_routine(), load_fragment(), main(), readout_thread(), receive_trigger_event(), register_equipment(), scan_fragment(), scheduler(), send_all_periodic_events(), send_event(), tr_start(), and tr_stop().

4.27.1.11 DWORD eqpmnt::last_called

Last time event was read

Definition at line 983 of file midas.h.

Referenced by scheduler(), and send_event().

4.27.1.12 DWORD eqpmnt::last_idle

Last time idle func was called

Definition at line 984 of file midas.h.

Referenced by scheduler().

4.27.1.13 char eqpmnt::name[NAME_LENGTH]

Equipment name

Definition at line 975 of file midas.h.

Referenced by display(), main(), register_equipment(), scheduler(), send_all_periodic_events(), send_event(), tr_start(), and tr_stop().

4.27.1.14 DWORD eqpmnt::odb_in

# updated ODB -> FE

Definition at line 992 of file midas.h.

Referenced by tr_start().

4.27.1.15 DWORD eqpmnt::odb_out

# updates FE -> ODB

Definition at line 991 of file midas.h.

Referenced by scheduler(), send_event(), and tr_start().

4.27.1.16 DWORD eqpmnt::poll_count

Needed to poll ’period’

Definition at line 985 of file midas.h.

Referenced by readout_thread(), register_equipment(), and scheduler().

4.27.1.17 INT(∗ eqpmnt::readout)(char ∗, INT)

Pointer to user readout routine

Referenced by interrupt_routine(), readout_thread(), scheduler(), and send_event().

4.27.1.18 DWORD eqpmnt::serial_number

event serial number

Definition at line 989 of file midas.h.

Referenced by eb_user(), interrupt_routine(), readout_thread(), scheduler(), send_event(), source_scan(), and tr_start().

4.27.1.19 EQUIPMENT_STATS eqpmnt::stats

Definition at line 995 of file midas.h.

Referenced by close_buffers(), display(), register_equipment(), scan_fragment(), scheduler(), send_event(), tr_start(), and tr_stop().

4.27.1.20 INT eqpmnt::status

One of FE_xxx

Definition at line 982 of file midas.h.

Referenced by display(), main(), register_equipment(), scheduler(), and send_all_periodic_events().

4.27.1.21 DWORD eqpmnt::subevent_number

subevent number

Definition at line 990 of file midas.h.

Referenced by receive_trigger_event(), scheduler(), and tr_start().

4.28 EQUIPMENT_INFO Struct Reference

Data Fields

- WORD event_id
- WORD trigger_mask
- char buffer [NAME_LENGTH]
- INT eq_type
- INT source
- char format [8]
- BOOL enabled
- INT read_on

- INT period
- double event_limit
- DWORD num_subevents
- INT history
- char frontend_host [NAME_LENGTH]
- char frontend_name [NAME_LENGTH]
- char frontend_file_name [256]

4.28.1 Field Documentation

4.28.1.1 char EQUIPMENT_INFO::buffer[NAME_LENGTH]

Event buffer to send events into

Definition at line 951 of file midas.h.

Referenced by register_equipment().

4.28.1.2 BOOL EQUIPMENT_INFO::enabled

Enable flag

Definition at line 955 of file midas.h.

Referenced by display(), register_equipment(), scheduler(), send_all_periodic_events(), and tr_start().

4.28.1.3 INT EQUIPMENT_INFO::eq_type

One of EQ_xxx

Definition at line 952 of file midas.h.

Referenced by main(), register_equipment(), scheduler(), send_event(), and tr_stop().

4.28.1.4 WORD EQUIPMENT_INFO::event_id

Event ID associated with equipm.

Definition at line 949 of file midas.h.

Referenced by interrupt_routine(), readout_thread(), register_equipment(), scheduler(), send_event(), and source_scan().

4.28.1.5 double EQUIPMENT_INFO::event_limit

Stop run when limit is reached

Definition at line 958 of file midas.h.

Referenced by register_equipment(), and scheduler().

4.28.1.6 char EQUIPMENT_INFO::format[8]

Data format to produce

Definition at line 954 of file midas.h.

Referenced by load_fragment(), and register_equipment().

4.28.1.7 char EQUIPMENT_INFO::frontend_file_name[256]

Source file used for user FE

Definition at line 963 of file midas.h.

Referenced by register_equipment().

4.28.1.8 char EQUIPMENT_INFO::frontend_host[NAME_LENGTH]

Host on which FE is running

Definition at line 961 of file midas.h.

Referenced by register_equipment().

4.28.1.9 char EQUIPMENT_INFO::frontend_name[NAME_LENGTH]

Frontend name

Definition at line 962 of file midas.h.

Referenced by register_equipment().

4.28.1.10 INT EQUIPMENT_INFO::history

Log history

Definition at line 960 of file midas.h.

Referenced by scheduler(), and send_event().

4.28.1.11 DWORD EQUIPMENT_INFO::num_subevents

Number of events in super event

Definition at line 959 of file midas.h.

Referenced by receive_trigger_event(), and scheduler().

4.28.1.12 INT EQUIPMENT_INFO::period

Readout interval/Polling time in ms

Definition at line 957 of file midas.h.

Referenced by register_equipment(), and scheduler().

4.28.1.13 INT EQUIPMENT_INFO::read_on

Combination of Read-On flags RO_xxx

Definition at line 956 of file midas.h.

Referenced by scheduler(), send_all_periodic_events(), and send_event().

4.28.1.14 INT EQUIPMENT_INFO::source

Event source (LAM/IRQ)

Definition at line 953 of file midas.h.

Referenced by main(), readout_thread(), register_equipment(), and scheduler().

4.28.1.15 WORD EQUIPMENT_INFO::trigger_mask

Trigger mask

Definition at line 950 of file midas.h.

Referenced by interrupt_routine(), readout_thread(), scheduler(), send_event(), and source_scan().

4.29 EQUIPMENT_STATS Struct Reference

4.29.1 Field Documentation

4.29.1.1 double EQUIPMENT_STATS::events_per_sec

Definition at line 968 of file midas.h.

Referenced by register_equipment(), scan_fragment(), and scheduler().

4.29.1.2 double EQUIPMENT_STATS::events_sent

Definition at line 967 of file midas.h.

Referenced by close_buffers(), display(), register_equipment(), scan_fragment(), scheduler(), send_event(), tr_start(), and tr_stop().

4.29.1.3 double EQUIPMENT_STATS::kbytes_per_sec

Definition at line 969 of file midas.h.

Referenced by register_equipment(), scan_fragment(), and scheduler().

4.30 EVENT_HEADER Struct Reference

4.30.1 Detailed Description

Event header

Definition at line 755 of file midas.h.

Data Fields

- short int event_id
- short int trigger_mask
- DWORD serial_number
- DWORD time_stamp
- DWORD data_size

4.30.2 Field Documentation

4.30.2.1 DWORD EVENT_HEADER::data_size

size of event in bytes w/o header

Definition at line 760 of file midas.h.

Referenced by bm_compose_event(), bm_convert_event_header(), bm_copy_from_cache(), bm_dispatch_from_cache(), bm_flush_cache(), bm_push_event(), bm_receive_event(), cm_msg(), cm_msg1(), eb_user(), interrupt_routine(), readout_thread(), receive_trigger_event(), scheduler(), and send_event().

4.30.2.2 short int EVENT_HEADER::event_id

event ID starting from one

Definition at line 756 of file midas.h.

Referenced by bm_compose_event(), bm_convert_event_header(), bm_dispatch_event(), bm_match_event(), interrupt_routine(), readout_thread(), scheduler(), and send_event().

4.30.2.3 DWORD EVENT_HEADER::serial_number

serial number starting from one

Definition at line 758 of file midas.h.

Referenced by bm_compose_event(), bm_convert_event_header(), eb_user(), interrupt_routine(), readout_thread(), receive_trigger_event(), scheduler(), and send_event().

4.30.2.4 DWORD EVENT_HEADER::time_stamp

time of production of event

Definition at line 759 of file midas.h.

Referenced by bm_compose_event(), bm_convert_event_header(), interrupt_routine(), readout_thread(), scheduler(), and send_event().

4.30.2.5 short int EVENT_HEADER::trigger_mask

hardware trigger mask

Definition at line 757 of file midas.h.

Referenced by bm_compose_event(), bm_convert_event_header(), bm_match_event(), interrupt_routine(), readout_thread(), scheduler(), and send_event().

4.31 EVENT_REQUEST Struct Reference

4.31.1 Detailed Description

Buffer structure

Definition at line 810 of file midas.h.

Data Fields

- INT id

- BOOL valid
- short int event_id
- short int trigger_mask
- INT sampling_type

4.31.2 Field Documentation

4.31.2.1 void(∗ EVENT_REQUEST::dispatch)(HNDLE, HNDLE, EVENT_HEADER ∗, void ∗)

4.31.2.2 short int EVENT_REQUEST::event_id

event ID

Definition at line 813 of file midas.h.

Referenced by bm_push_event(), bm_receive_event(), bm_send_event(), and bm_wait_for_free_space().

4.31.2.3 INT EVENT_REQUEST::id

request id

Definition at line 811 of file midas.h.

Referenced by bm_remove_event_request(), bm_send_event(), and bm_wait_for_free_space().

4.31.2.4 INT EVENT_REQUEST::sampling_type

dispatch function

Definition at line 815 of file midas.h.

Referenced by bm_remove_event_request(), bm_send_event(), and bm_wait_for_free_space().

4.31.2.5 short int EVENT_REQUEST::trigger_mask

trigger mask

Definition at line 814 of file midas.h.

Referenced by bm_push_event(), bm_receive_event(), bm_send_event(), and bm_wait_for_free_space().

4.31.2.6 BOOL EVENT_REQUEST::valid

indicating a valid entry

Definition at line 812 of file midas.h.

Referenced by bm_push_event(), bm_receive_event(), bm_remove_event_request(), bm_send_event(), and bm_wait_for_free_space().

4.32 EXP_PARAM Struct Reference

4.32.1 Field Documentation

4.32.1.1 char EXP_PARAM::comment[80]

Definition at line 27 of file experim.h.

Referenced by ana_end_of_run().

4.33 FREE_DESCRIP Struct Reference

Data Fields

- INT size
- INT next_free

4.33.1 Field Documentation

4.33.1.1 INT FREE_DESCRIP::next_free

Address of next free block

Definition at line 331 of file msystem.h.

Referenced by db_open_database().

4.33.1.2 INT FREE_DESCRIP::size

size in bytes

Definition at line 330 of file msystem.h.

Referenced by db_open_database().

4.34 GLOBAL_PARAM Struct Reference

4.34.1 Field Documentation

4.34.1.1 float GLOBAL_PARAM::adc_threshold

Definition at line 93 of file experim.h.

4.35 HIST_RECORD Struct Reference

4.35.1 Field Documentation

4.35.1.1 DWORD HIST_RECORD::data_size

Definition at line 1166 of file midas.h.

4.35.1.2 DWORD HIST_RECORD::def_offset

Definition at line 1165 of file midas.h.

4.35.1.3 DWORD HIST_RECORD::event_id

Definition at line 1163 of file midas.h.

4.35.1.4 DWORD HIST_RECORD::record_type

Definition at line 1162 of file midas.h.

4.35.1.5 DWORD HIST_RECORD::time

Definition at line 1164 of file midas.h.

4.36 HISTORY Struct Reference

4.36.1 Field Documentation

4.36.1.1 DWORD HISTORY::base_time

Definition at line 1189 of file midas.h.

4.36.1.2 DWORD HISTORY::def_fh

Definition at line 1188 of file midas.h.

4.36.1.3 DWORD HISTORY::def_offset

Definition at line 1190 of file midas.h.

4.36.1.4 DWORD HISTORY::event_id

Definition at line 1182 of file midas.h.

4.36.1.5 char HISTORY::event_name[NAME_LENGTH]

Definition at line 1183 of file midas.h.

4.36.1.6 DWORD HISTORY::hist_fh

Definition at line 1186 of file midas.h.

4.36.1.7 DWORD HISTORY::index_fh

Definition at line 1187 of file midas.h.

4.36.1.8 DWORD HISTORY::n_tag

Definition at line 1184 of file midas.h.

4.36.1.9 TAG∗ HISTORY::tag

Definition at line 1185 of file midas.h.

4.37 INDEX_RECORD Struct Reference

4.37.1 Field Documentation

4.37.1.1 DWORD INDEX_RECORD::event_id

Definition at line 1176 of file midas.h.

4.37.1.2 DWORD INDEX_RECORD::offset

Definition at line 1178 of file midas.h.

4.37.1.3 DWORD INDEX_RECORD::time

Definition at line 1177 of file midas.h.

4.38 KEY Struct Reference

Data Fields

- DWORD type
- INT num_values
- char name [NAME_LENGTH]
- INT data
- INT total_size
- INT item_size
- WORD access_mode
- WORD notify_count
- INT next_key
- INT parent_keylist
- INT last_written

4.38.1 Field Documentation

4.38.1.1 WORD KEY::access_mode

Access mode

Definition at line 886 of file midas.h.

Referenced by cm_cleanup(), db_create_key(), db_delete_key1(), db_find_key(), db_get_data(), db_get_data_index(), db_get_value(), db_open_database(), db_open_record(), db_set_data(), db_set_data_index(), and db_set_value().

4.38.1.2 INT KEY::data

Address of variable (offset)

Definition at line 883 of file midas.h.

Referenced by db_create_key(), db_delete_key1(), db_enum_key(), db_find_key(), db_get_data(), db_get_data_index(), db_get_key_info(), db_get_value(), db_open_database(), db_set_data(), db_set_data_index(), and db_set_value().

4.38.1.3 INT KEY::item_size

Size of single data item

Definition at line 885 of file midas.h.

Referenced by db_copy(), db_create_key(), db_get_data(), db_get_data_index(), db_get_key_info(), db_get_record(), db_get_record_size(), db_get_value(), db_open_database(), db_save_xml_key(), db_set_data(), db_set_data_index(), db_set_record(), db_set_value(), and update_odb().

4.38.1.4 INT KEY::last_written

Time of last write action

Definition at line 890 of file midas.h.

Referenced by db_get_key_time(), db_set_data(), db_set_data_index(), and db_set_value().

4.38.1.5 char KEY::name[NAME_LENGTH]

name of variable

Definition at line 882 of file midas.h.

Referenced by al_check(), al_reset_alarm(), cm_check_client(), cm_shutdown(), cm_transition(), db_check_record(), db_copy(), db_create_key(), db_find_key(), db_get_data(), db_get_data_index(), db_get_key_info(), db_get_record(), db_open_database(), db_save_struct(), db_save_xml_key(), db_set_data(), db_set_data_index(), db_set_record(), load_fragment(), and update_odb().

4.38.1.6 INT KEY::next_key

Address of next key

Definition at line 888 of file midas.h.

Referenced by db_create_key(), db_delete_key1(), db_enum_key(), and db_find_key().

4.38.1.7 WORD KEY::notify_count

Notify counter

Definition at line 887 of file midas.h.

Referenced by cm_cleanup(), db_delete_key1(), and db_open_database().

4.38.1.8 INT KEY::num_values

number of values

Definition at line 881 of file midas.h.

Referenced by cm_register_transition(), cm_transition(), db_check_record(), db_copy(), db_create_key(), db_get_data(), db_get_data_index(), db_get_key_info(), db_get_record(), db_get_record_size(), db_get_value(), db_open_database(), db_save_xml_key(), db_set_data(), db_set_data_index(), db_set_record(), db_set_value(), tr_start(), and update_odb().

4.38.1.9 INT KEY::parent_keylist

keylist to which this key belongs

Definition at line 889 of file midas.h.

Referenced by db_create_key(), db_delete_key1(), db_enum_key(), db_find_key(), and db_open_database().

4.38.1.10 INT KEY::total_size

Total size of data block

Definition at line 884 of file midas.h.

Referenced by db_copy(), db_create_key(), db_delete_key1(), db_open_database(), db_save_xml_key(), db_set_data(), db_set_data_index(), db_set_record(), db_set_value(), and tr_start().

4.38.1.11 DWORD KEY::type

TID_xxx type

Definition at line 880 of file midas.h.

Referenced by al_check(), cm_transition(), db_check_record(), db_copy(), db_create_key(), db_delete_key1(), db_enum_key(), db_find_key(), db_get_data(), db_get_data_index(), db_get_key(), db_get_key_info(), db_get_record(), db_get_record_size(), db_get_value(), db_open_database(), db_paste(), db_save_xml_key(), db_set_data(), db_set_data_index(), db_set_record(), db_set_value(), load_fragment(), and update_odb().

4.39 KEYLIST Struct Reference

Data Fields

- INT parent
- INT num_keys
- INT first_key

4.39.1 Field Documentation

4.39.1.1 INT KEYLIST::first_key

Address of first key

Definition at line 896 of file midas.h.

Referenced by db_create_key(), db_delete_key1(), db_enum_key(), db_find_key(), and db_open_database().

4.39.1.2 INT KEYLIST::num_keys

number of keys

Definition at line 895 of file midas.h.

Referenced by db_create_key(), db_delete_key1(), db_enum_key(), db_find_key(), db_get_key_info(), and db_open_database().

4.39.1.3 INT KEYLIST::parent

Address of parent key

Definition at line 894 of file midas.h.

Referenced by db_create_key(), db_delete_key1(), db_enum_key(), db_find_key(), and db_open_database().

4.40 MVME_INTERFACE Struct Reference

Data Fields

- int handle
- int index
- void ∗ info
- int am

- int dmode
- int blt_mode
- void ∗ table

4.40.1 Field Documentation

4.40.1.1 int MVME_INTERFACE::am

Address modifier

Definition at line 139 of file mvmestd.h.

4.40.1.2 int MVME_INTERFACE::blt_mode

Block transfer mode

Definition at line 141 of file mvmestd.h.

4.40.1.3 int MVME_INTERFACE::dmode

Data mode (D8,D16,D32,D64)

Definition at line 140 of file mvmestd.h.

4.40.1.4 int MVME_INTERFACE::handle

internal handle

Definition at line 136 of file mvmestd.h.

4.40.1.5 int MVME_INTERFACE::index

index of interface 0..n

Definition at line 137 of file mvmestd.h.

4.40.1.6 void∗ MVME_INTERFACE::info

internal info structure

Definition at line 138 of file mvmestd.h.

4.40.1.7 void* MVME_INTERFACE::table

Optional table for some drivers

Definition at line 142 of file mvmestd.h.

4.41 OPEN_RECORD Struct Reference

Data Fields

- INT handle
- WORD access_mode
- WORD flags

4.41.1 Field Documentation

4.41.1.1 WORD OPEN_RECORD::access_mode

R/W flags

Definition at line 336 of file msystem.h.

Referenced by cm_cleanup(), and db_open_database().

4.41.1.2 WORD OPEN_RECORD::flags

Data format, ...

Definition at line 337 of file msystem.h.

4.41.1.3 INT OPEN_RECORD::handle

Handle of record base key

Definition at line 335 of file msystem.h.

Referenced by cm_cleanup(), db_close_database(), and db_open_database().

4.42 PROGRAM_INFO Struct Reference

4.42.1 Detailed Description

Program information structure

Definition at line 1241 of file midas.h.

4.42.2 Field Documentation

4.42.2.1 char PROGRAM_INFO::alarm_class[32]

Definition at line 1249 of file midas.h.

Referenced by al_check().

4.42.2.2 BOOL PROGRAM_INFO::auto_restart

Definition at line 1248 of file midas.h.

Referenced by al_check().

4.42.2.3 BOOL PROGRAM_INFO::auto_start

Definition at line 1246 of file midas.h.

Referenced by cm_transition().

4.42.2.4 BOOL PROGRAM_INFO::auto_stop

Definition at line 1247 of file midas.h.

Referenced by cm_transition().

4.42.2.5 DWORD PROGRAM_INFO::check_interval

Definition at line 1244 of file midas.h.

Referenced by al_check().

4.42.2.6 DWORD PROGRAM_INFO::first_failed

Definition at line 1250 of file midas.h.

Referenced by al_check().

4.42.2.7 BOOL PROGRAM_INFO::required

Definition at line 1242 of file midas.h.

4.42.2.8 char PROGRAM_INFO::start_command[256]

Definition at line 1245 of file midas.h.

Referenced by al_check(), and cm_transition().

4.42.2.9 INT PROGRAM_INFO::watchdog_timeout

Definition at line 1243 of file midas.h.

4.43 RECORD_LIST Struct Reference

4.43.1 Field Documentation

4.43.1.1 WORD RECORD_LIST::access_mode

Definition at line 392 of file msystem.h.

Referenced by db_close_all_records(), db_close_record(), db_open_record(), db_send_changed_records(), and db_update_record().

4.43.1.2 INT RECORD_LIST::buf_size

Definition at line 395 of file msystem.h.

Referenced by db_open_record(), db_send_changed_records(), and db_update_record().

4.43.1.3 void∗ RECORD_LIST::copy

Definition at line 394 of file msystem.h.

Referenced by db_open_record(), and db_send_changed_records().

4.43.1.4 void∗ RECORD_LIST::data

Definition at line 393 of file msystem.h.

Referenced by db_open_record(), db_send_changed_records(), and db_update_record().

4.43.1.5 void(∗ RECORD_LIST::dispatcher)(INT, INT, void ∗)

Referenced by db_open_record(), and db_update_record().

4.43.1.6 HNDLE RECORD_LIST::handle

Definition at line 390 of file msystem.h.

Referenced by db_close_all_records(), db_close_record(), db_open_record(), and db_update_record().

4.43.1.7 HNDLE RECORD_LIST::hDB

Definition at line 391 of file msystem.h.

Referenced by db_close_record(), and db_open_record().

4.43.1.8 void∗ RECORD_LIST::info

Definition at line 397 of file msystem.h.

Referenced by db_open_record().

4.44 REQUEST_LIST Struct Reference

4.44.1 Field Documentation

4.44.1.1 INT REQUEST_LIST::buffer_handle

Definition at line 404 of file msystem.h.

Referenced by bm_close_buffer(), bm_dispatch_event(), and bm_request_event().

4.44.1.2 void(∗ REQUEST_LIST::dispatcher)(HNDLE, HNDLE, EVENT_HEADER ∗, void ∗)

Referenced by bm_dispatch_event(), and bm_request_event().

4.44.1.3 short int REQUEST_LIST::event_id

Definition at line 405 of file msystem.h.

Referenced by bm_request_event().

4.44.1.4 short int REQUEST_LIST::trigger_mask

Definition at line 406 of file msystem.h.

Referenced by bm_request_event().

4.45 RUNINFO Struct Reference

4.45.1 Detailed Description

Contains the main parameters regarding the run status. The containt reflects the current system ONLY if Midas clients are connected. Otherwise the status is erroneous.

Definition at line 1204 of file midas.h.

Data Fields

- INT state
- INT online_mode
- INT run_number
- INT transition_in_progress
- INT requested_transition
- char start_time [32]
- DWORD start_time_binary
- char stop_time [32]
- DWORD stop_time_binary

4.45.2 Field Documentation

4.45.2.1 INT RUNINFO::online_mode

Mode of operation online/offline

Definition at line 1206 of file midas.h.

Referenced by ana_end_of_run().

4.45.2.2 INT RUNINFO::requested_transition

Deferred transition request

Definition at line 1209 of file midas.h.

4.45.2.3 INT RUNINFO::run_number

Current processing run number

Definition at line 1207 of file midas.h.

Referenced by ana_end_of_run().

4.45.2.4 char RUNINFO::start_time[32]

ASCII of the last start time

Definition at line 1210 of file midas.h.

Referenced by ana_end_of_run().

4.45.2.5 DWORD RUNINFO::start_time_binary

Bin of the last start time

Definition at line 1211 of file midas.h.

4.45.2.6 INT RUNINFO::state

Current run condition

Definition at line 1205 of file midas.h.

4.45.2.7 char RUNINFO::stop_time[32]

ASCII of the last stop time

Definition at line 1212 of file midas.h.

4.45.2.8 DWORD RUNINFO::stop_time_binary

ASCII of the last stop time

Definition at line 1213 of file midas.h.

4.45.2.9 INT RUNINFO::transition_in_progress

Intermediate state during transition

Definition at line 1208 of file midas.h.

4.46 SCALER_COMMON Struct Reference

4.46.1 Field Documentation

4.46.1.1 char SCALER_COMMON::buffer[32]

Definition at line 181 of file experim.h.

4.46.1.2 BOOL SCALER_COMMON::enabled

Definition at line 185 of file experim.h.

4.46.1.3 WORD SCALER_COMMON::event_id

Definition at line 179 of file experim.h.

4.46.1.4 double SCALER_COMMON::event_limit

Definition at line 188 of file experim.h.

4.46.1.5 char SCALER_COMMON::format[8]

Definition at line 184 of file experim.h.

4.46.1.6 char SCALER_COMMON::frontend_file_name[256]

Definition at line 193 of file experim.h.

4.46.1.7 char SCALER_COMMON::frontend_host[32]

Definition at line 191 of file experim.h.

4.46.1.8 char SCALER_COMMON::frontend_name[32]

Definition at line 192 of file experim.h.

4.46.1.9 INT SCALER_COMMON::log_history

Definition at line 190 of file experim.h.

4.46.1.10 DWORD SCALER_COMMON::num_subevents

Definition at line 189 of file experim.h.

4.46.1.11 INT SCALER_COMMON::period

Definition at line 187 of file experim.h.

4.46.1.12 INT SCALER_COMMON::read_on

Definition at line 186 of file experim.h.

4.46.1.13 INT SCALER_COMMON::source

Definition at line 183 of file experim.h.

4.46.1.14 WORD SCALER_COMMON::trigger_mask

Definition at line 180 of file experim.h.

4.46.1.15 INT SCALER_COMMON::type

Definition at line 182 of file experim.h.

4.47 TAG Struct Reference

Data Fields

- char name [NAME_LENGTH]
- DWORD type
- DWORD n_data

4.47.1 Field Documentation

4.47.1.1 DWORD TAG::n_data

-

Definition at line 1029 of file midas.h.

Referenced by hs_dump().

4.47.1.2 char TAG::name[NAME_LENGTH]

-

Definition at line 1027 of file midas.h.

4.47.1.3 DWORD TAG::type

-

Definition at line 1028 of file midas.h.

Referenced by hs_dump().

4.48 TR_CLIENT Struct Reference

4.48.1 Detailed Description

dox***

Definition at line 2959 of file midas.c.

4.48.2 Field Documentation

4.48.2.1 char TR_CLIENT::client_name[NAME_LENGTH]

Definition at line 2962 of file midas.c.

4.48.2.2 char TR_CLIENT::host_name[HOST_NAME_LENGTH]

Definition at line 2961 of file midas.c.

4.48.2.3 int TR_CLIENT::port

Definition at line 2963 of file midas.c.

Referenced by cm_transition().

4.48.2.4 int TR_CLIENT::sequence_number

Definition at line 2960 of file midas.c.

Referenced by cm_transition().

4.49 TRIGGER_COMMON Struct Reference

4.49.1 Field Documentation

4.49.1.1 char TRIGGER_COMMON::buffer[32]

Definition at line 125 of file experim.h.

4.49.1.2 BOOL TRIGGER_COMMON::enabled

Definition at line 129 of file experim.h.

4.49.1.3 WORD TRIGGER_COMMON::event_id

Definition at line 123 of file experim.h.

4.49.1.4 double TRIGGER_COMMON::event_limit

Definition at line 132 of file experim.h.

4.49.1.5 char TRIGGER_COMMON::format[8]

Definition at line 128 of file experim.h.

4.49.1.6 char TRIGGER_COMMON::frontend_file_name[256]

Definition at line 137 of file experim.h.

4.49.1.7 char TRIGGER_COMMON::frontend_host[32]

Definition at line 135 of file experim.h.

4.49.1.8 char TRIGGER_COMMON::frontend_name[32]

Definition at line 136 of file experim.h.

4.49.1.9 INT TRIGGER_COMMON::log_history

Definition at line 134 of file experim.h.

4.49.1.10 DWORD TRIGGER_COMMON::num_subevents

Definition at line 133 of file experim.h.

4.49.1.11 INT TRIGGER_COMMON::period

Definition at line 131 of file experim.h.

4.49.1.12 INT TRIGGER_COMMON::read_on

Definition at line 130 of file experim.h.

4.49.1.13 INT TRIGGER_COMMON::source

Definition at line 127 of file experim.h.

4.49.1.14 WORD TRIGGER_COMMON::trigger_mask

Definition at line 124 of file experim.h.

4.49.1.15 INT TRIGGER_COMMON::type

Definition at line 126 of file experim.h.

4.50 TRIGGER_SETTINGS Struct Reference

4.50.1 Field Documentation

4.50.1.1 BYTE TRIGGER_SETTINGS::io506

Definition at line 163 of file experim.h.

5 Midas File Documentation

5.1 adccalib.c File Reference

5.1.1 Define Documentation

5.1.1.1 #define ADC_N_BINS 500

Definition at line 67 of file adccalib.c.

5.1.1.2 #define ADC_X_HIGH 4000

Definition at line 69 of file adccalib.c.

5.1.1.3 #define ADC_X_LOW 0

Definition at line 68 of file adccalib.c.

5.1.2 Function Documentation

5.1.2.1 INT adc_calib (EVENT_HEADER ∗, void ∗)

Definition at line 106 of file adccalib.c.

5.1.2.2 INT adc_calib_bor (INT *run_number*)

Definition at line 92 of file adccalib.c.

5.1.2.3 INT adc_calib_eor (INT *run_number*)

Definition at line 99 of file adccalib.c.

5.1.2.4 INT adc_calib_init (void)

Definition at line 71 of file adccalib.c.

5.1.2.5 ADC_CALIBRATION_PARAM_STR (adc_calibration_param_str)

5.1.3 Variable Documentation

5.1.3.1 ANA_MODULE adc_calib_module

Initial value:

```
 {
   "ADC calibration",
   "Stefan Ritt",
   adc_calib,
   adc_calib_bor,
   adc_calib_eor,
   adc_calib_init,
   NULL,
   &adccalib_param,
   sizeof(adccalib_param),
   adc_calibration_param_str,
}
```

Definition at line 48 of file adccalib.c.

5.1.3.2 ADC_CALIBRATION_PARAM adccalib_param

Definition at line 35 of file adccalib.c.

Referenced by adc_calib().

5.1.3.3 EXP_PARAM exp_param

Definition at line 51 of file analyzer.c.

Referenced by ana_end_of_run(), and analyzer_init().

5.1.3.4 TH1D* hAdcHists[N_ADC] `[static]`

Definition at line 63 of file adccalib.c.

Referenced by adc_calib(), and adc_calib_init().

5.1.3.5 RUNINFO runinfo

Definition at line 49 of file analyzer.c.

Referenced by ana_end_of_run(), and analyzer_init().

5.2 adcsum.c File Reference

5.2.1 Define Documentation

5.2.1.1 #define DEFINE_TESTS

Definition at line 22 of file adcsum.c.

5.2.1.2 #define PI 3.14159265359

Definition at line 33 of file adcsum.c.

5.2.2 Function Documentation

5.2.2.1 INT adc_summing (EVENT_HEADER ∗, void ∗)

Definition at line 87 of file adcsum.c.

5.2.2.2 INT adc_summing_bor (INT *run_number*)

5.2.2.3 INT adc_summing_init (void)

Definition at line 72 of file adcsum.c.

5.2.2.4 ADC_SUMMING_PARAM_STR (adc_summing_param_str)

5.2.2.5 DEF_TEST (high_sum)

5.2.2.6 DEF_TEST (low_sum)

5.2.3 Variable Documentation

5.2.3.1 ANA_MODULE adc_summing_module

Initial value:

```
 {
   "ADC summing",
   "Stefan Ritt",
   adc_summing,
   NULL,
   NULL,
   adc_summing_init,
   NULL,
   &adc_summing_param,
   sizeof(adc_summing_param),
   adc_summing_param_str,
}
```

Definition at line 53 of file adcsum.c.

5.2.3.2 ADC_SUMMING_PARAM adc_summing_param

Definition at line 38 of file adcsum.c.

Referenced by adc_summing().

5.2.3.3 TH1D ∗ hAdcAvg `[static]`

Definition at line 68 of file adcsum.c.

Referenced by adc_summing(), and adc_summing_init().

5.2.3.4 TH1D∗ hAdcSum `[static]`

Definition at line 68 of file adcsum.c.

Referenced by adc_summing(), and adc_summing_init().

5.3 alarm.c File Reference

5.3.1 Detailed Description

The Midas Alarm file

Definition in file alarm.c.

Functions

- INT al_trigger_alarm (char ∗alarm_name, char ∗alarm_message, char ∗default_class, char ∗cond_str, INT type)
- INT al_reset_alarm (char ∗alarm_name)
- INT al_check ()

5.4 analyzer.c File Reference

5.4.1 Function Documentation

5.4.1.1 INT ana_begin_of_run (INT *run_number*, char ∗ *error*)

Definition at line 199 of file analyzer.c.

5.4.1.2 INT ana_end_of_run (INT *run_number*, char ∗ *error*)

Definition at line 206 of file analyzer.c.

5.4.1.3 INT ana_pause_run (INT *run_number*, char ∗ *error*)

Definition at line 263 of file analyzer.c.

5.4.1.4 INT ana_resume_run (INT *run_number*, char ∗ *error*)

Definition at line 270 of file analyzer.c.

5.4.1.5 INT analyzer_exit ()

Definition at line 192 of file analyzer.c.

5.4.1.6 INT analyzer_init ()

Definition at line 137 of file analyzer.c.

5.4.1.7 INT analyzer_loop ()

Definition at line 277 of file analyzer.c.

5.4.1.8 ASUM_BANK_STR (asum_bank_str)

5.4.2 Variable Documentation

5.4.2.1 ANA_MODULE adc_calib_module

Definition at line 48 of file adccalib.c.

5.4.2.2 ANA_MODULE adc_summing_module

Definition at line 53 of file adcsum.c.

5.4.2.3 BANK_LIST ana_scaler_bank_list[ ]

Initial value:

```
 {

   {"SCLR", TID_DWORD, N_ADC, NULL},


   {"ACUM", TID_DOUBLE, N_ADC, NULL},
   {""},
}
```

Definition at line 88 of file analyzer.c.

5.4.2.4 BANK_LIST ana_trigger_bank_list[ ]

Initial value:

```
 {


   {"ADC0", TID_WORD, N_ADC, NULL},
   {"TDC0", TID_WORD, N_TDC, NULL},


   {"CADC", TID_FLOAT, N_ADC, NULL},
   {"ASUM", TID_STRUCT, sizeof(ASUM_BANK), asum_bank_str},

   {""},
}
```

Definition at line 75 of file analyzer.c.

5.4.2.5 ANALYZE_REQUEST analyze_request[ ]

Definition at line 99 of file analyzer.c.

5.4.2.6 INT analyzer_loop_period = 0

Definition at line 43 of file analyzer.c.

5.4.2.7 char* analyzer_name = "Analyzer"

Definition at line 40 of file analyzer.c.

Referenced by analyzer_init().

5.4.2.8 EXP_PARAM exp_param

Definition at line 51 of file analyzer.c.

Referenced by ana_end_of_run(), and analyzer_init().

5.4.2.9 GLOBAL_PARAM global_param

Definition at line 50 of file analyzer.c.

5.4.2.10 INT odb_size = DEFAULT_ODB_SIZE

Definition at line 46 of file analyzer.c.

Referenced by cm_connect_experiment1().

5.4.2.11 RUNINFO runinfo

Definition at line 49 of file analyzer.c.

Referenced by ana_end_of_run(), and analyzer_init().

5.4.2.12 ANA_MODULE scaler_accum_module

Definition at line 32 of file scaler.c.

5.4.2.13 ANA_MODULE∗ scaler_module[ ]

Initial value:

```
 {
   &scaler_accum_module,
   NULL
}
```

Definition at line 60 of file analyzer.c.

5.4.2.14 ANA_MODULE∗ trigger_module[ ]

Initial value:

```
 {
   &adc_calib_module,
   &adc_summing_module,
   NULL
}
```

Definition at line 65 of file analyzer.c.

5.4.2.15 TRIGGER_SETTINGS trigger_settings

Definition at line 52 of file analyzer.c.

5.5 analyzer.dox File Reference

5.6 appendixA.dox File Reference

5.7 appendixB.dox File Reference

5.8 appendixC.dox File Reference

5.9 appendixD.dox File Reference

5.10 appendixE.dox File Reference

5.11 appendixG.dox File Reference

5.12 components.dox File Reference

5.13 ebuser.c File Reference

5.13.1 Detailed Description

The Event builder user file

Definition in file ebuser.c.

Functions

- INT eb_begin_of_run (INT, char ∗, char ∗)
- INT eb_end_of_run (INT, char ∗)
- INT eb_user (INT nfrag, BOOL mismatch, EBUILDER_CHANNEL ∗ebch, EVENT_HEADER ∗pheader, void ∗pevent, INT ∗dest_size)

Variables

- INT lModulo = 100

5.13.2 Function Documentation

5.13.2.1 INT eb_begin_of_run (INT *rn*, char ∗ *UserField*, char ∗ *error*)

Hook to the event builder task at PreStart transition.

Parameters:

rn run number

UserField argument from /Ebuilder/Settings

error error string to be passed back to the system.

Returns:

EB_SUCCESS

Referenced by tr_start().

5.13.2.2 INT eb_end_of_run (INT *rn*, char ∗ *error*)

Hook to the event builder task at completion of event collection after receiving the Stop transition.

Parameters:

rn run number

error error string to be passed back to the system.

Returns:

EB_SUCCESS

Referenced by close_buffers().

5.13.2.3 INT eb_user (INT *nfrag*, BOOL *mismatch*, EBUILDER_CHANNEL ∗ *ebch*, EVENT_HEADER ∗ *pheader*, void ∗ *pevent*, INT ∗ *dest_size*)

Hook to the event builder task after the reception of all fragments of the same serial number. The destination event has already the final EVENT_HEADER setup with the data size set to 0. It is than possible to add private data at this point using the proper bank calls.

The ebch[] array structure points to nfragment channel structure with the following content:

```
typedef struct {
    char  name[32];         // Fragment name (Buffer name).
    DWORD serial;           // Serial fragment number.
    char *pfragment;        // Pointer to fragment (EVENT_HEADER *)
    ...
} EBUILDER_CHANNEL;
```

The correct code for including your own MIDAS bank is shown below where **TID_xxx** is one of the valid Bank type starting with **TID_** for midas format or **xxx_BKTYPE** for Ybos data format. **bank_name** is a 4 character descriptor. **pdata** has to be declared accordingly with the bank type. Refers to the ebuser.c source code for further description.

It is not possible to mix within the same destination event different event format!

```
  // Event is empty, fill it with BANK_HEADER
  // If you need to add your own bank at this stage

  bk_init(pevent);
  bk_create(pevent, bank_name, TID_xxxx, &pdata);
  *pdata++ = ...;
  *dest_size = bk_close(pevent, pdata);
  pheader->data_size = *dest_size + sizeof(EVENT_HEADER);
```

For YBOS format, use the following example.

```
  ybk_init(pevent);
  ybk_create(pevent, "EBBK", I4_BKTYPE, &pdata);
  *pdata++ = 0x12345678;
  *pdata++ = 0x87654321;
  *dest_size = ybk_close(pevent, pdata);
  *dest_size *= 4;
  pheader->data_size = *dest_size + sizeof(YBOS_BANK_HEADER);
```

Parameters:

nfrag Number of fragment.

mismatch Midas Serial number mismatch flag.

ebch Structure to all the fragments.

pheader Destination pointer to the header.

pevent Destination pointer to the bank header.

dest_size Destination event size in bytes.

Returns:

EB_SUCCESS

Definition at line 187 of file ebuser.c.

Referenced by source_scan().

5.13.2.4 INT ebuilder_exit ()

5.13.2.5 INT ebuilder_init ()

5.13.2.6 INT ebuilder_loop ()

5.13.2.7 INT ebuser (INT, BOOL *mismatch*, EBUILDER_CHANNEL ∗, EVENT_HEADER ∗, void ∗, INT ∗)

5.13.2.8 INT read_scaler_event (char ∗ *pevent*, INT *off*)

Definition at line 341 of file frontend.c.

5.13.3 Variable Documentation

5.13.3.1 BOOL debug

Definition at line 68 of file mfe.c.

5.13.3.2 INT display_period = 3000

Definition at line 32 of file ebuser.c.

5.13.3.3 EBUILDER_SETTINGS ebset

Definition at line 29 of file mevb.c.

Referenced by eb_user(), handFlush(), main(), source_booking(), source_scan(), and tr_start().

5.13.3.4 BOOL ebuilder_call_loop = FALSE

Definition at line 29 of file ebuser.c.

5.13.3.5 EQUIPMENT equipment[ ]

Initial value:

```
 {
   {"EB",
    {1, 1,
     "SYSTEM",
     0,
     0,
     "MIDAS",
     TRUE,
     },
    },

  {""}
}
```

Definition at line 59 of file ebuser.c.

5.13.3.6 INT event_buffer_size = 10 ∗ 10000

Definition at line 41 of file ebuser.c.

5.13.3.7 char∗ frontend_file_name = __FILE__

Definition at line 26 of file ebuser.c.

5.13.3.8 char∗ frontend_name = "Ebuilder"

Definition at line 23 of file ebuser.c.

5.13.3.9 INT lModulo = 100

Global var for testing passed at BOR.

Globals

Definition at line 45 of file ebuser.c.

Referenced by eb_begin_of_run().

5.13.3.10 INT max_event_size = 10000

Definition at line 35 of file ebuser.c.

5.13.3.11 INT max_event_size_frag = 5 ∗ 1024 ∗ 1024

Definition at line 38 of file ebuser.c.

5.14 elog.c File Reference

Functions

- INT el_submit (int run, char ∗author, char ∗type, char ∗syst, char ∗subject, char ∗text, char ∗reply_to, char ∗encoding, char ∗afilename1, char ∗buffer1, INT buffer_size1, char ∗afilename2, char ∗buffer2, INT buffer_size2, char ∗afilename3, char ∗buffer3, INT buffer_size3, char ∗tag, INT tag_size)

5.15 esone.c File Reference

5.15.1 Detailed Description

The ESONE CAMAC standard call file

Definition in file esone.c.

Functions

- INLINE void ccinit (void)
- INLINE int fccinit (void)
- INLINE void cdreg (int ∗ext, const int b, const int c, const int n, const int a)
- INLINE void cssa (const int f, int ext, unsigned short ∗d, int ∗q)
- INLINE void cfsa (const int f, const int ext, unsigned long ∗d, int ∗q)
- INLINE void cccc (const int ext)
- INLINE void cccz (const int ext)
- INLINE void ccci (const int ext, int l)
- INLINE void ctci (const int ext, int ∗l)
- INLINE void cccd (const int ext, int l)

- INLINE void ctcd (const int ext, int ∗l)
- INLINE void cdlam (int ∗lam, const int b, const int c, const int n, const int a, const int inta[2])
- INLINE void ctgl (const int ext, int ∗l)
- INLINE void cclm (const int lam, int l)
- INLINE void cclnk (const int lam, void(∗isr)(void))
- INLINE void cculk (const int lam)
- INLINE void ccrgl (const int lam)
- INLINE void cclc (const int lam)
- INLINE void ctlm (const int lam, int ∗l)
- INLINE void cfga (int f[ ], int exta[ ], int intc[ ], int qa[ ], int cb[ ])
- INLINE void csga (int f[ ], int exta[ ], int intc[ ], int qa[ ], int cb[ ])
- INLINE void cfmad (int f, int extb[ ], int intc[ ], int cb[ ])
- INLINE void csmad (int f, int extb[ ], int intc[ ], int cb[ ])
- INLINE void cfubc (const int f, int ext, int intc[ ], int cb[ ])
- INLINE void csubc (const int f, int ext, int intc[ ], int cb[ ])
- INLINE void cfubr (const int f, int ext, int intc[ ], int cb[ ])
- INLINE void csubr (const int f, int ext, int intc[ ], int cb[ ])

5.15.2 Function Documentation

5.15.2.1 INLINE void cccc (const int *ext*)

Control Crate Clear.

Generate Crate Clear function. Execute cam_crate_clear()

Parameters:

ext external address

Returns:

void

Definition at line 193 of file esone.c.

5.15.2.2 INLINE void cccd (const int *ext*, int *l*)

Control Crate D.

Enable or Disable Crate Demand.

Parameters:

ext external address

l action l=0 -> Clear D, l=1 -> Set D

Returns:

void

Definition at line 267 of file esone.c.

5.15.2.3 INLINE void ccci (const int *ext*, int *l*)

Control Crate I.

Set or Clear Dataway Inhibit, Execute cam_inhinit_set() /clear()

Parameters:

ext external address

l action l=0 -> Clear I, l=1 -> Set I

Returns:

void

Definition at line 228 of file esone.c.

5.15.2.4 INLINE void cccz (const int *ext*)

Control Crate Z.

Generate Dataway Initialize. Execute cam_crate_zinit()

Parameters:

ext external address

Returns:

void

Definition at line 210 of file esone.c.

5.15.2.5 INLINE void ccinit (void)

CAMAC initialization

CAMAC initialization must be called before any other ESONE subroutine call.

Returns:

void

Definition at line 72 of file esone.c.

5.15.2.6 INLINE void cclc (const int *lam*)

Control Clear LAM.

Clear the LAM of the station pointer by the lam address.

Parameters:

lam external address

Returns:

void

Definition at line 428 of file esone.c.

5.15.2.7 INLINE void cclm (const int *lam*, int *l*)

Control Crate LAM.

Enable or Disable LAM. Execute F24 for disable, F26 for enable.

Parameters:

lam external address

l action l=0 -> disable LAM , l=1 -> enable LAM

Returns:

void

Definition at line 348 of file esone.c.

5.15.2.8 INLINE void cclnk (const int *lam*, void(∗)(void) *isr*)

Link LAM to service procedure

Link a specific service routine to a LAM. Since this routine is executed asynchronously, care must be taken on re-entrancy.

Parameters:

lam external address

isr name of service procedure

Returns:

void

Definition at line 371 of file esone.c.

5.15.2.9 INLINE void ccrgl (const int *lam*)

Relink LAM

Re-enable LAM in the controller

Parameters:

lam external address

Returns:

void

Definition at line 408 of file esone.c.

5.15.2.10 INLINE void cculk (const int *lam*)

Unlink LAM from service procedure

Performs complementary operation to cclnk.

Parameters:

lam external address

Returns:

void

Definition at line 391 of file esone.c.

5.15.2.11 INLINE void cdlam (int ∗ *lam*, const int *b*, const int *c*, const int *n*, const int *a*, const int *inta*[2])

Control Declare LAM.

Declare LAM, Identical to cdreg.

Parameters:

lam external LAM address

b branch number (0..7)

c crate number (0..)

n station number (0..30)

a sub-address (0..15)

inta implementation dependent

Returns:

void

Definition at line 311 of file esone.c.

5.15.2.12 INLINE void cdreg (int ∗ *ext*, const int *b*, const int *c*, const int *n*, const int *a*)

Control Declaration REGister.

Compose an external address from BCNA for later use. Accessing CAMAC through ext could be faster if the external address is memory mapped to the processor (hardware dependent). Some CAMAC controller do not have this option see Supported hardware.

Parameters:

ext external address

b branch number (0..7)

c crate number (0..)

n station number (0..30)

a sub-address (0..15)

Returns:

void

Definition at line 109 of file esone.c.

Referenced by cdlam().

5.15.2.13 INLINE void cfga (int *f*[ ], int *exta*[ ], int *intc*[ ], int *qa*[ ], int *cb*[ ])

Control Full (24bit) word General Action.

Parameters:

f function code

exta[ ] external address array

intc[ ] data array

qa[ ] Q response array

cb[ ] control block array

cb[0] : number of function to perform
cb[1] : returned number of function performed

Returns:

void

Definition at line 467 of file esone.c.

5.15.2.14 INLINE void cfmad (int *f*, int *extb*[ ], int *intc*[ ], int *cb*[ ])

Control Full (24bit) Address Q scan.

Scan all sub-address while Q=1 from a0..a15 max from address extb[0] and store corresponding data in intc[]. If Q=0 while A<15 or A=15 then cross station boundary is applied (n-> n+1) and sub-address is reset (a=0). Perform action until either cb[0] action are performed or current external address exceeds extb[1].

implementation of cb[2] for LAM recognition is not implemented.

Parameters:

f function code

extb[ ] external address array

extb[0] : first valid external address
extb[1] : last valid external address

intc[ ] data array

cb[ ] control block array

cb[0] : number of function to perform
cb[1] : returned number of function performed

Returns:

void

Definition at line 521 of file esone.c.

5.15.2.15 INLINE void cfsa (const int *f*, const int *ext*, unsigned long ∗ *d*, int ∗ *q*)

Control Full Operation.

24 bit operation on a given external CAMAC address.

The range of the f is hardware dependent. The number indicated below are for standard ANSI/IEEE Std (758-1979) Execute cam24i for f<8, cam24o for f>15, camc_q for (f>7 or f>23)

Parameters:

f function code (0..31)

ext external address

d data long word

q Q response

Returns:

void

Definition at line 165 of file esone.c.

Referenced by cfga(), cfubc(), and cfubr().

5.15.2.16 INLINE void cfubc (const int *f*, int *ext*, int *intc*[ ], int *cb*[ ])

Control Full (24bit) Block Repeat with Q-stop.

Execute function f on address ext with data intc[] while Q.

Parameters:

f function code

ext external address array

intc[ ] data array

cb[ ] control block array

cb[0] : number of function to perform
cb[1] : returned number of function performed

Returns:

void

Definition at line 618 of file esone.c.

5.15.2.17 INLINE void cfubr (const int *f*, int *ext*, int *intc*[ ], int *cb*[ ])

Repeat Mode Block Transfer (24bit).

Execute function f on address ext with data intc[] if Q. If noQ keep current intc[] data. Repeat cb[0] times.

Parameters:

f function code

ext external address array

intc[ ] data array

cb[ ] control block array

cb[0] : number of function to perform
cb[1] : returned number of function performed

Returns:

void

Definition at line 681 of file esone.c.

5.15.2.18 INLINE void csga (int *f*[ ], int *exta*[ ], int *intc*[ ], int *qa*[ ], int *cb*[ ])

Control (16bit) word General Action.

Parameters:

f function code

exta[ ] external address array

intc[ ] data array

qa[ ] Q response array

cb[ ] control block array

cb[0] : number of function to perform
cb[1] : returned number of function performed

Returns:

void

Definition at line 490 of file esone.c.

5.15.2.19 INLINE void csmad (int *f*, int *extb*[ ], int *intc*[ ], int *cb*[ ])

Control (16bit) Address Q scan.

Scan all sub-address while Q=1 from a0..a15 max from address extb[0] and store corresponding data in intc[]. If Q=0 while A<15 or A=15 then cross station boundary is applied (n-> n+1) and sub-address is reset (a=0). Perform action until either cb[0] action are performed or current external address exceeds extb[1].

implementation of cb[2] for LAM recognition is not implemented.

Parameters:

f function code

extb[ ] external address array

extb[0] : first valid external address
extb[1] : last valid external address

intc[ ] data array

cb[ ] control block array

cb[0] : number of function to perform
cb[1] : returned number of function performed

Returns:

void

Definition at line 573 of file esone.c.

5.15.2.20 INLINE void cssa (const int *f*, int *ext*, unsigned short ∗ *d*, int ∗ *q*)

Control Short Operation.

16 bit operation on a given external CAMAC address.

The range of the f is hardware dependent. The number indicated below are for standard ANSI/IEEE Std (758-1979) Execute cam16i for f<8, cam16o for f>15, camc_q for (f>7 or f>23)

Parameters:

- ***f*** function code (0..31)
- ***ext*** external address
- ***d*** data word
- ***q*** Q response

Returns:

void

Definition at line 130 of file esone.c.

Referenced by csga(), csubc(), and csubr().

5.15.2.21 INLINE void csubc (const int *f*, int *ext*, int *intc*[ ], int *cb*[ ])

Control (16bit) Block Repeat with Q-stop.

Execute function f on address ext with data intc[] while Q.

Parameters:

- ***f*** function code
- ***ext*** external address array
- ***intc[ ]*** data array
- ***cb[ ]*** control block array
  - cb[0] : number of function to perform
  - cb[1] : returned number of function performed

Returns:

void

Definition at line 649 of file esone.c.

5.15.2.22 INLINE void csubr (const int *f*, int *ext*, int *intc*[ ], int *cb*[ ])

Repeat Mode Block Transfer (16bit).

Execute function f on address ext with data intc[] if Q. If noQ keep current intc[] data. Repeat cb[0] times.

Parameters:

- ***f*** function code

ext external address array

intc[ ] data array

cb[ ] control block array
  cb[0] : number of function to perform
  cb[1] : returned number of function performed

Returns:
void

Definition at line 712 of file esone.c.

5.15.2.23 INLINE void ctcd (const int *ext*, int $*$ *l*)

Control Test Crate D.

Test Crate Demand.

Parameters:

ext external address

l D cleared -> l=0, D set -> l=1

Returns:
void

Definition at line 289 of file esone.c.

5.15.2.24 INLINE void ctci (const int *ext*, int $*$ *l*)

Test Crate I.

Test Crate Inhibit, Execute cam_inhibit_test()

Parameters:

ext external address

l action l=0 -> Clear I, l=1 -> Set I

Returns:
void

Definition at line 249 of file esone.c.

5.15.2.25 INLINE void ctgl (const int *ext*, int ∗ *l*)

Control Test Demand Present.

Test the LAM register.

Parameters:

- ***ext*** external LAM register address
- ***l*** 1 !=0 if any LAM is set.

Returns:

void

Definition at line 328 of file esone.c.

5.15.2.26 INLINE void ctlm (const int *lam*, int ∗ *l*)

Test LAM.

Test the LAM of the station pointed by lam. Performs an F8

Parameters:

- ***lam*** external address
- ***l*** No LAM-> l=0, LAM present-> l=1

Returns:

void

Definition at line 446 of file esone.c.

5.15.2.27 INLINE int fccinit (void)

CAMAC initialization with return status

fccinit can be called instead of ccinit to determine if the initialization was successful

Returns:

1 for success, 0 for failure

Definition at line 86 of file esone.c.

5.16 eventbuilder.dox File Reference

5.17 experim.h File Reference

Data Structures

- struct EXP_PARAM
- struct ADC_CALIBRATION_PARAM
- struct ADC_SUMMING_PARAM
- struct GLOBAL_PARAM
- struct ASUM_BANK
- struct TRIGGER_COMMON
- struct TRIGGER_SETTINGS
- struct SCALER_COMMON

5.17.1 Define Documentation

5.17.1.1 #define ADC_CALIBRATION_PARAM_DEFINED

Definition at line 38 of file experim.h.

5.17.1.2 #define ADC_CALIBRATION_PARAM_STR(_name)

Value:

```
char *_name[] = {\
"[.]",\
"Pedestal = INT[8] :",\
"[0] 174",\
"[1] 194",\
"[2] 176",\
"[3] 182",\
"[4] 185",\
"[5] 215",\
"[6] 202",\
"[7] 202",\
"Software Gain = FLOAT[8] :",\
"[0] 1",\
"[1] 1",\
"[2] 1",\
"[3] 1",\
"[4] 1",\
"[5] 1",\
"[6] 1",\
"[7] 1",\
"Histo threshold = DOUBLE : 20",\
"",\
NULL }
```

Definition at line 46 of file experim.h.

5.17.1.3 #define ADC_SUMMING_PARAM_DEFINED

Definition at line 74 of file experim.h.

5.17.1.4 #define ADC_SUMMING_PARAM_STR(_name)

Value:

```
char *_name[] = {\
"[.]",\
"ADC threshold = FLOAT : 5",\
"",\
NULL }
```

Definition at line 80 of file experim.h.

5.17.1.5 #define ASUM_BANK_DEFINED

Definition at line 106 of file experim.h.

5.17.1.6 #define ASUM_BANK_STR(_name)

Value:

```
char *_name[] = {\
"[.]",\
"Sum = FLOAT : 0",\
"Average = FLOAT : 0",\
"",\
NULL }
```

Definition at line 113 of file experim.h.

5.17.1.7 #define EXP_PARAM_DEFINED

Definition at line 24 of file experim.h.

5.17.1.8 #define EXP_PARAM_STR(_name)

Value:

```
char *_name[] = {\
"[.]",\
"Comment = STRING : [80] Test",\
"",\
NULL }
```

Definition at line 30 of file experim.h.

Referenced by analyzer_init().

5.17.1.9 #define GLOBAL_PARAM_DEFINED

Definition at line 90 of file experim.h.

5.17.1.10 #define GLOBAL_PARAM_STR(_name)

Value:

```
char *_name[] = {\
"[.]",\
"ADC Threshold = FLOAT : 5",\
"",\
NULL }
```

Definition at line 96 of file experim.h.

Referenced by analyzer_init().

5.17.1.11 #define SCALER_COMMON_DEFINED

Definition at line 176 of file experim.h.

5.17.1.12 #define SCALER_COMMON_STR(_name)

Value:

```
char *_name[] = {\
"[.]",\
"Event ID = WORD : 2",\
"Trigger mask = WORD : 0",\
"Buffer = STRING : [32] SYSTEM",\
"Type = INT : 17",\
"Source = INT : 0",\
"Format = STRING : [8] MIDAS",\
"Enabled = BOOL : y",\
"Read on = INT : 377",\
"Period = INT : 10000",\
"Event limit = DOUBLE : 0",\
"Num subevents = DWORD : 0",\
"Log history = INT : 0",\
"Frontend host = STRING : [32] pc810",\
"Frontend name = STRING : [32] Sample Frontend",\
"Frontend file name = STRING : [256] C:\Midas\examples\experiment\frontend.c",\
"",\
NULL }
```

Definition at line 196 of file experim.h.

5.17.1.13 #define TRIGGER_COMMON_DEFINED

Definition at line 120 of file experim.h.

5.17.1.14 #define TRIGGER_COMMON_STR(_name)

Value:

```
char *_name[] = {\
"[.]",\
"Event ID = WORD : 1",\
"Trigger mask = WORD : 0",\
"Buffer = STRING : [32] SYSTEM",\
"Type = INT : 2",\
"Source = INT : 16777215",\
"Format = STRING : [8] MIDAS",\
"Enabled = BOOL : y",\
"Read on = INT : 257",\
"Period = INT : 500",\
"Event limit = DOUBLE : 0",\
"Num subevents = DWORD : 0",\
"Log history = INT : 0",\
"Frontend host = STRING : [32] pc810",\
"Frontend name = STRING : [32] Sample Frontend",\
"Frontend file name = STRING : [256] C:\Midas\examples\experiment\frontend.c",\
"",\
NULL }
```

Definition at line 140 of file experim.h.

5.17.1.15 #define TRIGGER_SETTINGS_DEFINED

Definition at line 160 of file experim.h.

5.17.1.16 #define TRIGGER_SETTINGS_STR(_name)

Value:

```
char *_name[] = {\
"[.]",\
"IO506 = BYTE : 7",\
"",\
NULL }
```

Definition at line 166 of file experim.h.

Referenced by analyzer_init().

5.18 frontend.c File Reference

5.18.1 Define Documentation

5.18.1.1 #define CRATE 0

Definition at line 56 of file frontend.c.

Referenced by frontend_init(), read_scaler_event(), and read_trigger_event().

5.18.1.2 #define N_ADC 4

Definition at line 51 of file frontend.c.

Referenced by adc_calib().

5.18.1.3 #define N_SCLR 4

Definition at line 53 of file frontend.c.

5.18.1.4 #define N_TDC 4

Definition at line 52 of file frontend.c.

5.18.1.5 #define SLOT_ADC 1

Definition at line 58 of file frontend.c.

Referenced by read_trigger_event().

5.18.1.6 #define SLOT_IO 23

Definition at line 57 of file frontend.c.

Referenced by frontend_init(), and read_trigger_event().

5.18.1.7 #define SLOT_SCLR 3

Definition at line 60 of file frontend.c.

Referenced by read_scaler_event().

5.18.1.8 #define SLOT_TDC 2

Definition at line 59 of file frontend.c.

Referenced by read_trigger_event().

5.18.2 Function Documentation

5.18.2.1 INT begin_of_run (INT *run_number*, char ∗ *error*)

Referenced by tr_start().

5.18.2.2 INT end_of_run (INT *run_number*, char ∗ *error*)

Referenced by tr_stop().

5.18.2.3 INT frontend_exit ()

5.18.2.4 INT frontend_init ()

5.18.2.5 INT frontend_loop ()

5.18.2.6 INT interrupt_configure (INT *cmd*, INT *source*, POINTER_T *adr*)

Definition at line 254 of file frontend.c.

Referenced by main(), readout_enable(), and register_equipment().

5.18.2.7 INT pause_run (INT *run_number*, char ∗ *error*)

Referenced by tr_pause().

5.18.2.8 INT poll_event (INT *source*, INT *count*, BOOL *test*)

Definition at line 233 of file frontend.c.

Referenced by readout_thread(), register_equipment(), and scheduler().

5.18.2.9 INT read_scaler_event (char ∗ *pevent*, INT *off*)

5.18.2.10 INT read_trigger_event (char ∗ *pevent*, INT *off*)

Definition at line 271 of file frontend.c.

5.18.2.11 void register_cnaf_callback (int *debug*)

Referenced by frontend_init().

5.18.2.12 INT resume_run (INT *run_number*, char $*$ *error*)

Referenced by tr_resume().

5.18.3 Variable Documentation

5.18.3.1 INT display_period = 3000

Definition at line 39 of file frontend.c.

5.18.3.2 EQUIPMENT equipment[ ]

Definition at line 79 of file frontend.c.

5.18.3.3 INT event_buffer_size = 100 $*$ 10000

Definition at line 48 of file frontend.c.

5.18.3.4 BOOL frontend_call_loop = FALSE

Definition at line 36 of file frontend.c.

5.18.3.5 char$*$ frontend_file_name = __FILE__

Definition at line 33 of file frontend.c.

5.18.3.6 char$*$ frontend_name = "Sample Frontend"

Definition at line 31 of file frontend.c.

5.18.3.7 INT max_event_size = 10000

Definition at line 42 of file frontend.c.

5.18.3.8 INT max_event_size_frag = 5 $*$ 1024 $*$ 1024

Definition at line 45 of file frontend.c.

5.19 history.c File Reference

Functions

- INT hs_set_path (char ∗path)
- INT hs_dump (DWORD event_id, DWORD start_time, DWORD end_time, DWORD interval, BOOL binary_time)

5.20 internal.dox File Reference

5.21 introduction.dox File Reference

5.22 Makefile File Reference

5.22.1 Function Documentation

5.22.1.1 do install v D m $$file (PREFIX)

5.22.1.2 do install v D m $$file (SYSINC_DIR)

5.22.1.3 do install v D m $$file (SYSBIN_DIR)

5.22.1.4 then echo Making directory (BIN_DIR)

5.22.1.5 then echo Making directory (LIB_DIR)

5.22.1.6 then echo Making directory (OS_DIR)

Referenced by cm_connect_experiment1(), and cm_scan_experiments().

5.22.1.7 fi ln fs (SYSBIN_DIR)

5.22.1.8 done ifeq ($(OSTYPE), linux)

5.22.1.9 do install v m (BIN_DIR)

5.22.1.10 done ifdef CERNLIB install v m (LIB_DIR)

5.22.1.11 done install v m (UTL_DIR)

5.22.1.12 mkdir (BIN_DIR)

5.22.1.13 mkdir (LIB_DIR)

5.22.1.14 mkdir (OS_DIR)

5.22.1.15 then chmod s (SYSBIN_DIR)

Referenced by db_update_record().

5.22.1.16 fi echo echo No utilities install to (SYSBIN_DIR)

Definition at line 554 of file Makefile.

5.22.1.17 done echo echo Installing library and objects to (SYSLIB_DIR)

5.22.2 Variable Documentation

5.22.2.1 LIB_DIR __pad0__

Definition at line 309 of file Makefile.

5.22.2.2 BIN_DIR __pad1__

Definition at line 315 of file Makefile.

5.22.2.3 ifdef NEED_MYSQL CFLAGS

Definition at line 325 of file Makefile.

5.22.2.4 fi

Definition at line 318 of file Makefile.

5.22.2.5 fi if

Definition at line 550 of file Makefile.

Referenced by cm_cleanup().

5.22.2.6 done minimal_install

Definition at line 540 of file Makefile.

5.22.2.7 OSTYPE

Definition at line 33 of file Makefile.

5.23 mcstd.h File Reference

5.23.1 Detailed Description

The Midas CAMAC include file

Definition in file mcstd.h.

Functions

- EXTERNAL INLINE void EXPRT cam16i (const int c, const int n, const int a, const int f, WORD ∗d)
- EXTERNAL INLINE void EXPRT cam24i (const int c, const int n, const int a, const int f, DWORD ∗d)
- EXTERNAL INLINE void EXPRT cam8i_q (const int c, const int n, const int a, const int f, BYTE ∗d, int ∗x, int ∗q)
- EXTERNAL INLINE void EXPRT cam16i_q (const int c, const int n, const int a, const int f, WORD ∗d, int ∗x, int ∗q)
- EXTERNAL INLINE void EXPRT cam24i_q (const int c, const int n, const int a, const int f, DWORD ∗d, int ∗x, int ∗q)
- EXTERNAL INLINE void EXPRT cam16i_r (const int c, const int n, const int a, const int f, WORD ∗∗d, const int r)
- EXTERNAL INLINE void EXPRT cam24i_r (const int c, const int n, const int a, const int f, DWORD ∗∗d, const int r)
- EXTERNAL INLINE void EXPRT cam8i_rq (const int c, const int n, const int a, const int f, BYTE ∗∗d, const int r)
- EXTERNAL INLINE void EXPRT cam16i_rq (const int c, const int n, const int a, const int f, WORD ∗∗d, const int r)
- EXTERNAL INLINE void EXPRT cam24i_rq (const int c, const int n, const int a, const int f, DWORD ∗∗d, const int r)
- EXTERNAL INLINE void EXPRT cam8i_sa (const int c, const int n, const int a, const int f, BYTE ∗∗d, const int r)
- EXTERNAL INLINE void EXPRT cam16i_sa (const int c, const int n, const int a, const int f, WORD ∗∗d, const int r)
- EXTERNAL INLINE void EXPRT cam24i_sa (const int c, const int n, const int a, const int f, DWORD ∗∗d, const int r)
- EXTERNAL INLINE void EXPRT cam8i_sn (const int c, const int n, const int a, const int f, BYTE ∗∗d, const int r)
- EXTERNAL INLINE void EXPRT cam16i_sn (const int c, const int n, const int a, const int f, WORD ∗∗d, const int r)
- EXTERNAL INLINE void EXPRT cam24i_sn (const int c, const int n, const int a, const int f, DWORD ∗∗d, const int r)
- EXTERNAL INLINE void EXPRT cami (const int c, const int n, const int a, const int f, WORD ∗d)
- EXTERNAL INLINE void EXPRT cam8o (const int c, const int n, const int a, const int f, BYTE d)
- EXTERNAL INLINE void EXPRT cam16o (const int c, const int n, const int a, const int f, WORD d)
- EXTERNAL INLINE void EXPRT cam24o (const int c, const int n, const int a, const int f, DWORD d)
- EXTERNAL INLINE void EXPRT cam8o_q (const int c, const int n, const int a, const int f, BYTE d, int ∗x, int ∗q)

- EXTERNAL INLINE void EXPRT cam16o_q (const int c, const int n, const int a, const int f, WORD d, int ∗x, int ∗q)
- EXTERNAL INLINE void EXPRT cam24o_q (const int c, const int n, const int a, const int f, DWORD d, int ∗x, int ∗q)
- EXTERNAL INLINE void EXPRT cam8o_r (const int c, const int n, const int a, const int f, BYTE ∗d, const int r)
- EXTERNAL INLINE void EXPRT cam16o_r (const int c, const int n, const int a, const int f, WORD ∗d, const int r)
- EXTERNAL INLINE void EXPRT cam24o_r (const int c, const int n, const int a, const int f, DWORD ∗d, const int r)
- EXTERNAL INLINE void EXPRT camo (const int c, const int n, const int a, const int f, WORD d)
- EXTERNAL INLINE int EXPRT camc_chk (const int c)
- EXTERNAL INLINE void EXPRT camc (const int c, const int n, const int a, const int f)
- EXTERNAL INLINE void EXPRT camc_q (const int c, const int n, const int a, const int f, int ∗q)
- EXTERNAL INLINE void EXPRT camc_sa (const int c, const int n, const int a, const int f, const int r)
- EXTERNAL INLINE void EXPRT camc_sn (const int c, const int n, const int a, const int f, const int r)
- EXTERNAL INLINE int EXPRT cam_init (void)
- EXTERNAL INLINE int EXPRT cam_init_rpc (char ∗host_name, char ∗exp_name, char ∗fe_name, char ∗client_name, char ∗rpc_server)
- EXTERNAL INLINE void EXPRT cam_exit (void)
- EXTERNAL INLINE void EXPRT cam_inhibit_set (const int c)
- EXTERNAL INLINE void EXPRT cam_inhibit_clear (const int c)
- EXTERNAL INLINE int EXPRT cam_inhibit_test (const int c)
- EXTERNAL INLINE void EXPRT cam_crate_clear (const int c)
- EXTERNAL INLINE void EXPRT cam_crate_zinit (const int c)
- EXTERNAL INLINE void EXPRT cam_lam_enable (const int c, const int n)
- EXTERNAL INLINE void EXPRT cam_lam_disable (const int c, const int n)
- EXTERNAL void cam_lam_read (const int c, DWORD ∗lam)
- EXTERNAL INLINE void EXPRT cam_lam_clear (const int c, const int n)
- EXTERNAL INLINE int EXPRT cam_lam_wait (int ∗c, DWORD ∗n, const int millisec)
- EXTERNAL INLINE void EXPRT cam_interrupt_enable (const int c)
- EXTERNAL INLINE void EXPRT cam_interrupt_disable (const int c)
- EXTERNAL INLINE int EXPRT cam_interrupt_test (const int c)
- EXTERNAL INLINE void EXPRT cam_interrupt_attach (const int c, const int n, void(∗isr)(void))
- EXTERNAL INLINE void EXPRT cam_interrupt_detach (const int c, const int n)

5.24 mevb.c File Reference

Defines

- #define SERVER_CACHE_SIZE 100000

Functions

- INT source_scan (INT fmt, EQUIPMENT_INFO ∗eq_info)
- INT eb_begin_of_run (INT, char ∗, char ∗)
- INT eb_end_of_run (INT, char ∗)
- INT eb_user (INT, BOOL mismatch, EBUILDER_CHANNEL ∗, EVENT_HEADER ∗, void ∗, INT ∗)

5.24.1 Define Documentation

5.24.1.1 #define DEFAULT_FE_TIMEOUT 60000

Definition at line 27 of file mevb.c.

5.24.1.2 #define EQUIPMENT_COMMON_STR "\Event ID = WORD : 0\n\Trigger mask = WORD : 0\n\Buffer = STRING : [32] SYSTEM\n\Type = INT : 0\n\Source = INT : 0\n\Format = STRING : [8] FIXED\n\Enabled = BOOL : 0\n\Read on = INT : 0\n\Period = INT : 0\n\Event limit = DOUBLE : 0\n\Num subevents = DWORD : 0\n\Log history = INT : 0\n\Frontend host = STRING : [32] \n\Frontend name = STRING : [32] \n\Frontend file name = STRING : [256] \n\"

Definition at line 82 of file mevb.c.

5.24.1.3 #define EQUIPMENT_STATISTICS_STR "\Events sent = DOUBLE : 0\n\Events per sec. = DOUBLE : 0\n\kBytes per sec. = DOUBLE : 0\n\"

Definition at line 100 of file mevb.c.

5.24.1.4 #define ODB_UPDATE_TIME 1000

Definition at line 25 of file mevb.c.

5.24.1.5 #define SERVER_CACHE_SIZE 100000

dox∗∗

Definition at line 23 of file mevb.c.

5.24.2 Function Documentation

5.24.2.1 INT close_buffers (void)

Definition at line 832 of file mevb.c.

Referenced by scan_fragment().

5.24.2.2 INT eb_begin_of_run (INT *rn*, char ∗ *UserField*, char ∗ *error*)

Hook to the event builder task at PreStart transition.

Parameters:

rn run number

UserField argument from /Ebuilder/Settings

error error string to be passed back to the system.

Returns:

EB_SUCCESS

Definition at line 105 of file ebuser.c.

5.24.2.3 INT eb_end_of_run (INT *rn*, char ∗ *error*)

Hook to the event builder task at completion of event collection after receiving the Stop transition.

Parameters:

rn run number

error error string to be passed back to the system.

Returns:

EB_SUCCESS

Definition at line 120 of file ebuser.c.

5.24.2.4 INT eb_mfragment_add (char ∗ *pdest*, char ∗ *psrce*, INT ∗ *size*)

Definition at line 422 of file mevb.c.

5.24.2.5 INT eb_user (INT *nfrag*, BOOL *mismatch*, EBUILDER_CHANNEL ∗ *ebch*, EVENT_HEADER ∗ *pheader*, void ∗ *pevent*, INT ∗ *dest_size*)

Hook to the event builder task after the reception of all fragments of the same serial number. The destination event has already the final EVENT_HEADER setup with the data size set to 0. It is than possible to add private data at this point using the proper bank calls.

The ebch[] array structure points to nfragment channel structure with the following content:

```
typedef struct {
    char  name[32];         // Fragment name (Buffer name).
    DWORD serial;           // Serial fragment number.
    char *pfragment;        // Pointer to fragment (EVENT_HEADER *)
    ...
} EBUILDER_CHANNEL;
```

The correct code for including your own MIDAS bank is shown below where **TID_xxx** is one of the valid Bank type starting with **TID_** for midas format or **xxx_BKTYPE** for Ybos data format. **bank_name** is a 4 character descriptor. **pdata** has to be declared accordingly with the bank type. Refers to the ebuser.c source code for further description.

It is not possible to mix within the same destination event different event format!

```
  // Event is empty, fill it with BANK_HEADER
  // If you need to add your own bank at this stage

  bk_init(pevent);
  bk_create(pevent, bank_name, TID_xxxx, &pdata);
  *pdata++ = ...;
  *dest_size = bk_close(pevent, pdata);
  pheader->data_size = *dest_size + sizeof(EVENT_HEADER);
```

For YBOS format, use the following example.

```
  ybk_init(pevent);
  ybk_create(pevent, "EBBK", I4_BKTYPE, &pdata);
  *pdata++ = 0x12345678;
  *pdata++ = 0x87654321;
  *dest_size = ybk_close(pevent, pdata);
  *dest_size *= 4;
  pheader->data_size = *dest_size + sizeof(YBOS_BANK_HEADER);
```

Parameters:

nfrag Number of fragment.

mismatch Midas Serial number mismatch flag.

ebch Structure to all the fragments.

pheader Destination pointer to the header.

pevent Destination pointer to the bank header.

dest_size Destination event size in bytes.

Returns:

EB_SUCCESS

Definition at line 187 of file ebuser.c.

Referenced by source_scan().

5.24.2.6 INT eb_yfragment_add (char ∗ *pdest*, char ∗ *psrce*, INT ∗ *size*)

Definition at line 478 of file mevb.c.

5.24.2.7 INT ebuilder_exit (void)

Definition at line 86 of file ebuser.c.

Referenced by main().

5.24.2.8 INT ebuilder_init (void)

Definition at line 80 of file ebuser.c.

Referenced by main().

5.24.2.9 INT ebuilder_loop (void)

Definition at line 92 of file ebuser.c.

5.24.2.10 void free_event_buffer (INT *nfrag*)

Definition at line 680 of file mevb.c.

Referenced by main(), source_booking(), and source_unbooking().

5.24.2.11 INT handFlush (void)

Definition at line 692 of file mevb.c.

Referenced by close_buffers().

5.24.2.12 INT load_fragment (void)

Definition at line 231 of file mevb.c.

Referenced by main().

5.24.2.13 int main (int *argc*, char ∗∗ *argv*)

Definition at line 1040 of file mevb.c.

5.24.2.14 INT register_equipment (void)

Definition at line 107 of file mevb.c.

5.24.2.15 INT scan_fragment (void)

Definition at line 310 of file mevb.c.

Referenced by main().

5.24.2.16 INT source_booking (void)

Definition at line 726 of file mevb.c.

Referenced by tr_start().

5.24.2.17 INT source_scan (INT *fmt*, EQUIPMENT_INFO ∗ *eq_info*)

Scan all the fragment source once per call.

1. This will retrieve the full midas event not swapped (except the MIDAS_HEADER) for each fragment if possible. The fragment will be stored in the channel event pointer.
2. if after a full nfrag path some frag are still not cellected, it returns with the frag# missing for timeout check.
3. If ALL fragments are present it will check the midas serial# for a full match across all the fragments.
4. If the serial check fails it returns with "event mismatch" and will abort the event builder but not stop the run for now.
5. If the serial check is passed, it will call the user_build function where the destination event is going to be composed.

Parameters:

fmt Fragment format type

eq_info Equipement pointer

Returns:

EB_NO_MORE_EVENT, EB_COMPOSE_TIMEOUT if different then SUCCESS (bm_compose, rpc_sent error)

Definition at line 881 of file mevb.c.

Referenced by scan_fragment().

5.24.2.18 INT source_unbooking (void)

Definition at line 798 of file mevb.c.

Referenced by close_buffers(), and main().

5.24.2.19 INT tr_start (INT *rn*, char ∗ *error*)

Definition at line 550 of file mevb.c.

5.24.2.20 INT tr_stop (INT *rn*, char ∗ *error*)

Definition at line 667 of file mevb.c.

5.24.2.21 INT ybos_event_swap (DWORD ∗ *pevt*)

Referenced by eb_yfragment_add(), and source_scan().

5.24.3 Variable Documentation

5.24.3.1 BOOL abort_requested = FALSE stop_requested = TRUE

Definition at line 50 of file mevb.c.

Referenced by close_buffers(), scan_fragment(), and tr_start().

5.24.3.2 DWORD actual_millitime

Definition at line 36 of file mevb.c.

5.24.3.3 DWORD actual_time

Definition at line 35 of file mevb.c.

5.24.3.4 char bars[ ] = "|\\-/"

Definition at line 48 of file mevb.c.

Referenced by scan_fragment().

5.24.3.5 char buffer_name[NAME_LENGTH]

Definition at line 41 of file mevb.c.

Referenced by bm_open_buffer(), bm_push_event(), load_fragment(), and main().

5.24.3.6 BOOL debug = FALSE debug1 = FALSE

Definition at line 45 of file mevb.c.

5.24.3.7 char* dest_event

Definition at line 43 of file mevb.c.

Referenced by load_fragment(), and source_scan().

5.24.3.8 INT display_period

Definition at line 32 of file ebuser.c.

5.24.3.9 EBUILDER_CHANNEL ebch[MAX_CHANNELS]

Definition at line 30 of file mevb.c.

Referenced by eb_user(), free_event_buffer(), handFlush(), load_fragment(), main(), scan_fragment(), source_booking(), source_scan(), and source_unbooking().

5.24.3.10 EBUILDER_SETTINGS ebset

Definition at line 29 of file mevb.c.

Referenced by eb_user(), handFlush(), main(), source_booking(), source_scan(), and tr_start().

5.24.3.11 EQUIPMENT equipment[ ]

Definition at line 59 of file ebuser.c.

5.24.3.12 INT event_buffer_size

Definition at line 41 of file ebuser.c.

5.24.3.13 char expt_name[NAME_LENGTH]

Definition at line 40 of file mevb.c.

Referenced by main().

5.24.3.14 BOOL frontend_call_loop

Definition at line 36 of file frontend.c.

5.24.3.15 char∗ frontend_file_name

Definition at line 26 of file ebuser.c.

5.24.3.16 char∗ frontend_name

Definition at line 23 of file ebuser.c.

5.24.3.17 HNDLE hDB

Definition at line 44 of file mevb.c.

5.24.3.18 HNDLE hEqKey

Definition at line 44 of file mevb.c.

Referenced by load_fragment().

5.24.3.19 HNDLE hESetKey

Definition at line 44 of file mevb.c.

5.24.3.20 HNDLE hKey

Definition at line 44 of file mevb.c.

Referenced by analyzer_init(), cm_connect_client(), cm_delete_client_info(), cm_deregister_transition(), cm_disconnect_experiment(), cm_exist(), cm_get_client_info(), cm_msg_log(), cm_msg_log1(), cm_msg_retrieve(), cm_register_deferred_transition(), cm_register_transition(), cm_set_client_info(), cm_set_transition_sequence(), cm_set_watchdog_params(), cm_shutdown(), cm_transition(), db_check_record(), db_close_record(), db_copy(), db_copy_xml(), db_create_key(), db_create_link(), db_create_record(), db_delete_key(), db_delete_key1(), db_enum_key(), db_find_key(), db_get_data(), db_get_data_index(), db_get_key(), db_get_key_info(), db_get_key_time(), db_get_record(), db_get_record_size(), db_open_record(), db_paste(), db_paste_node(), db_save(), db_save_struct(), db_save_xml(), db_save_xml_key(), db_set_data(), db_set_data_index(), db_set_record(), db_set_value(), db_update_record(), device_driver(), logger_root(), main(), register_equipment(), tr_start(), and update_odb().

5.24.3.21 char host_name[HOST_NAME_LENGTH]

Definition at line 39 of file mevb.c.

5.24.3.22 HNDLE hStatKey

Definition at line 44 of file mevb.c.

5.24.3.23 HNDLE hSubkey

Definition at line 44 of file mevb.c.

Referenced by cm_connect_client(), cm_exist(), cm_set_client_info(), cm_shutdown(), cm_transition(), db_copy(), db_create_record(), db_save_xml_key(), and load_fragment().

5.24.3.24 int i_bar

Definition at line 49 of file mevb.c.

Referenced by scan_fragment().

5.24.3.25 DWORD last_time

Definition at line 34 of file mevb.c.

Referenced by scan_fragment().

5.24.3.26 INT max_event_size

Definition at line 35 of file ebuser.c.

5.24.3.27 INT max_event_size_frag

Definition at line 38 of file ebuser.c.

5.24.3.28 INT(∗ meb_fragment_add)(char ∗, char ∗, INT ∗)

Definition at line 53 of file mevb.c.

Referenced by load_fragment(), and source_scan().

5.24.3.29 char mevb_svn_revision[ ] = "$Id: mevb.c 3659 2007-04-03 14:30:48Z ritt@PSI.CH $"

Definition at line 38 of file mevb.c.

Referenced by main().

5.24.3.30 INT nfragment

Definition at line 42 of file mevb.c.

Referenced by load_fragment(), source_booking(), source_scan(), and source_unbooking().

5.24.3.31 INT run_number

Definition at line 33 of file mevb.c.

5.24.3.32 INT run_state

Definition at line 32 of file mevb.c.

5.24.3.33 DWORD stop_time = 0 request_stop_time = 0

Definition at line 51 of file mevb.c.

Referenced by close_buffers().

5.24.3.34 BOOL wheel = FALSE

Definition at line 47 of file mevb.c.

Referenced by main().

5.25 mfe.c File Reference

5.25.1 Define Documentation

5.25.1.1 #define DEFAULT_FE_TIMEOUT 60000

Definition at line 54 of file mfe.c.

Referenced by main().

5.25.1.2 #define EQUIPMENT_COMMON_STR "\Event ID = WORD : 0\n\Trigger mask = WORD : 0\n\Buffer = STRING : [32] SYSTEM\n\Type = INT : 0\n\Source = INT : 0\n\Format = STRING : [8] FIXED\n\Enabled = BOOL : 0\n\Read on = INT : 0\n\Period = INT : 0\n\Event limit = DOUBLE : 0\n\Num subevents = DWORD : 0\n\Log history = INT : 0\n\Frontend host = STRING : [32] \n\Frontend name = STRING : [32] \n\Frontend file name = STRING : [256] \n\"

Definition at line 117 of file mfe.c.

Referenced by register_equipment().

5.25.1.3 #define EQUIPMENT_STATISTICS_STR "\Events sent = DOUBLE : 0\n\Events per sec. = DOUBLE : 0\n\kBytes per sec. = DOUBLE : 0\n\"

Definition at line 135 of file mfe.c.

Referenced by register_equipment().

5.25.1.4 #define ODB_UPDATE_TIME 1000

Definition at line 52 of file mfe.c.

Referenced by scheduler().

5.25.1.5 #define SERVER_CACHE_SIZE 100000

Definition at line 50 of file mfe.c.

Referenced by register_equipment().

5.25.2 Function Documentation

5.25.2.1 INT begin_of_run (INT *run_number*, char $*$ *error*)

Definition at line 186 of file frontend.c.

5.25.2.2 void display (BOOL *bInit*)

Definition at line 1447 of file mfe.c.

Referenced by main(), and scheduler().

5.25.2.3 INT end_of_run (INT *run_number*, char $*$ *error*)

Definition at line 195 of file frontend.c.

5.25.2.4 INT frontend_exit (void)

Definition at line 179 of file frontend.c.

Referenced by main().

5.25.2.5 INT frontend_init (void)

Definition at line 151 of file frontend.c.

Referenced by main().

5.25.2.6 INT frontend_loop (void)

Definition at line 216 of file frontend.c.

Referenced by scheduler().

5.25.2.7 INT get_frontend_index ()

Definition at line 2097 of file mfe.c.

5.25.2.8 INT interrupt_configure (INT *cmd*, INT *source*, POINTER_T *adr*)

Definition at line 254 of file frontend.c.

Referenced by main(), readout_enable(), and register_equipment().

5.25.2.9 void interrupt_routine (void)

Definition at line 1269 of file mfe.c.

Referenced by register_equipment().

5.25.2.10 BOOL logger_root ()

Definition at line 1520 of file mfe.c.

Referenced by receive_trigger_event(), and scheduler().

5.25.2.11 int main (int *argc*, char ∗ *argv*[ ])

Definition at line 2107 of file mfe.c.

5.25.2.12 INT manual_trigger (INT *idx*, void ∗ *prpc_param*[ ])

Definition at line 282 of file mfe.c.

Referenced by register_equipment().

5.25.2.13 int message_print (const char ∗ *msg*)

Definition at line 1429 of file mfe.c.

Referenced by main().

5.25.2.14 INT pause_run (INT *run_number*, char ∗ *error*)

Definition at line 202 of file frontend.c.

5.25.2.15 INT poll_event (INT *source*, INT *count*, BOOL *test*)

Definition at line 233 of file frontend.c.

Referenced by readout_thread(), register_equipment(), and scheduler().

5.25.2.16 void readout_enable (BOOL *flag*)

Definition at line 1249 of file mfe.c.

Referenced by main(), register_equipment(), scheduler(), tr_pause(), tr_resume(), tr_start(), and tr_stop().

5.25.2.17 int readout_enabled (void)

Definition at line 1244 of file mfe.c.

Referenced by readout_thread(), and scheduler().

5.25.2.18 int readout_thread (void ∗ *param*)

Definition at line 1305 of file mfe.c.

Referenced by register_equipment().

5.25.2.19 int receive_trigger_event (EQUIPMENT ∗ *eq*)

Definition at line 1379 of file mfe.c.

Referenced by scheduler(), and tr_stop().

5.25.2.20 INT register_equipment (void)

Definition at line 519 of file mfe.c.

Referenced by main().

5.25.2.21 INT resume_run (INT *run_number*, char ∗ *error*)

Definition at line 209 of file frontend.c.

5.25.2.22 int sc_thread (void ∗ *info*)

Definition at line 293 of file mfe.c.

Referenced by device_driver().

5.25.2.23 INT scheduler (void)

Definition at line 1547 of file mfe.c.

Referenced by main().

5.25.2.24 void send_all_periodic_events (INT *transition*)

Definition at line 1216 of file mfe.c.

Referenced by tr_pause(), tr_resume(), tr_start(), and tr_stop().

5.25.2.25 int send_event (INT *idx*)

Definition at line 1057 of file mfe.c.

Referenced by scheduler(), and send_all_periodic_events().

5.25.2.26 INT tr_pause (INT *rn*, char ∗ *error*)

Definition at line 233 of file mfe.c.

Referenced by main().

5.25.2.27 INT tr_resume (INT *rn*, char ∗ *error*)

Definition at line 258 of file mfe.c.

Referenced by main().

5.25.2.28 INT tr_start (INT *rn*, char ∗ *error*)

Definition at line 145 of file mfe.c.

Referenced by main().

5.25.2.29 INT tr_stop (INT *rn*, char ∗ *error*)

Definition at line 179 of file mfe.c.

Referenced by main().

5.25.2.30 void update_odb (EVENT_HEADER ∗ *pevent*, HNDLE *hKey*, INT *format*)

Definition at line 909 of file mfe.c.

Referenced by receive_trigger_event(), scheduler(), and send_event().

5.25.3 Variable Documentation

5.25.3.1 int _readout_enabled_flag = 0 `[static]`

Definition at line 1242 of file mfe.c.

Referenced by readout_enable().

5.25.3.2 DWORD actual_millitime

Definition at line 59 of file mfe.c.

Referenced by scan_fragment(), and scheduler().

5.25.3.3 DWORD actual_time

Definition at line 58 of file mfe.c.

Referenced by scheduler().

5.25.3.4 DWORD auto_restart = 0

Definition at line 69 of file mfe.c.

Referenced by scheduler().

5.25.3.5 BOOL debug

Definition at line 68 of file mfe.c.

Referenced by main().

5.25.3.6 INT display_period

Definition at line 32 of file ebuser.c.

Referenced by scheduler().

5.25.3.7 EQUIPMENT equipment[ ]

Definition at line 59 of file ebuser.c.

Referenced by close_buffers(), display(), load_fragment(), main(), register_equipment(), scan_fragment(), scheduler(), send_all_periodic_events(), send_event(), source_scan(), tr_start(), and tr_stop().

5.25.3.8 void* event_buffer

Definition at line 97 of file mfe.c.

Referenced by main().

5.25.3.9 INT event_buffer_size

Definition at line 41 of file ebuser.c.

Referenced by main(), and register_equipment().

5.25.3.10 char exp_name[NAME_LENGTH]

Definition at line 62 of file mfe.c.

Referenced by cm_connect_experiment(), cm_connect_experiment1(), cm_get_environment(), cm_list_experiments(), cm_select_experiment(), and main().

5.25.3.11 INT fe_stop = 0

Definition at line 67 of file mfe.c.

5.25.3.12 void* frag_buffer = NULL

Definition at line 98 of file mfe.c.

Referenced by register_equipment().

5.25.3.13 BOOL frontend_call_loop

Definition at line 36 of file frontend.c.

5.25.3.14 char∗ frontend_file_name

Definition at line 26 of file ebuser.c.

Referenced by register_equipment().

5.25.3.15 INT frontend_index = -1

Definition at line 71 of file mfe.c.

Referenced by main(), and register_equipment().

5.25.3.16 char∗ frontend_name

Definition at line 23 of file ebuser.c.

Referenced by load_fragment(), main(), register_equipment(), scan_fragment(), source_scan(), tr_start(), and tr_stop().

5.25.3.17 char full_frontend_name[256]

Definition at line 63 of file mfe.c.

Referenced by display(), main(), and register_equipment().

5.25.3.18 HNDLE hDB

Definition at line 73 of file mfe.c.

Referenced by al_check(), al_reset_alarm(), al_trigger_alarm(), ana_end_of_run(), analyzer_init(), bm_open_buffer(), cm_check_client(), cm_connect_client(), cm_connect_experiment1(), cm_delete_client_info(), cm_deregister_transition(), cm_disconnect_experiment(), cm_exist(), cm_get_client_info(), cm_get_experiment_database(), cm_get_watchdog_info(), cm_msg_log(), cm_msg_log1(), cm_msg_retrieve(), cm_register_deferred_transition(), cm_register_transition(), cm_set_client_info(), cm_set_transition_sequence(), cm_set_watchdog_params(), cm_shutdown(), cm_transition(), db_check_record(), db_close_database(), db_close_record(), db_copy(), db_copy_xml(), db_create_key(), db_create_link(), db_create_record(), db_delete_key(), db_delete_key1(), db_enum_key(), db_find_key(), db_get_data(), db_get_data_index(), db_get_key(), db_get_key_info(), db_get_key_time(), db_get_record(), db_get_record_size(), db_get_value(), db_load(), db_lock_database(), db_open_database(), db_open_record(), db_paste(), db_paste_node(), db_paste_xml(), db_protect_database(), db_save(), db_save_struct(), db_save_xml(), db_save_xml_key(), db_send_changed_records(), db_set_data(), db_set_data_index(), db_set_record(), db_set_value(), db_set_value_index(), db_unlock_database(), db_update_record(), el_submit(), load_fragment(), logger_root(), main(), register_equipment(), scheduler(), tr_start(), and update_odb().

5.25.3.19 char host_name[HOST_NAME_LENGTH]

Definition at line 61 of file mfe.c.

Referenced by cm_connect_client(), cm_connect_experiment(), cm_connect_experiment1(), cm_get_environment(), cm_list_experiments(), cm_select_experiment(), cm_set_client_info(), cm_transition(), display(), and main().

5.25.3.20 EQUIPMENT* interrupt_eq = NULL

Definition at line 94 of file mfe.c.

Referenced by interrupt_routine(), main(), register_equipment(), and scheduler().

5.25.3.21 INT manual_trigger_event_id = 0

Definition at line 70 of file mfe.c.

Referenced by manual_trigger(), and scheduler().

5.25.3.22 INT max_bytes_per_sec

Definition at line 65 of file mfe.c.

Referenced by scheduler().

5.25.3.23 INT max_event_size

Definition at line 35 of file ebuser.c.

Referenced by load_fragment(), main(), rb_create(), readout_thread(), register_equipment(), scheduler(), send_event(), and source_booking().

5.25.3.24 INT max_event_size_frag

Definition at line 38 of file ebuser.c.

Referenced by main(), register_equipment(), scheduler(), and send_event().

5.25.3.25 EQUIPMENT* multithread_eq = NULL

Definition at line 95 of file mfe.c.

Referenced by readout_thread(), and register_equipment().

5.25.3.26 INT optimize = 0

Definition at line 66 of file mfe.c.

5.25.3.27 int rbh1 = 0 rbh2=0 rbh1_next=0 rbh2_next=0

Definition at line 101 of file mfe.c.

Referenced by interrupt_routine(), readout_thread(), and register_equipment().

5.25.3.28 volatile int readout_thread_active = 0

Definition at line 104 of file mfe.c.

Referenced by readout_thread().

5.25.3.29 INT rpc_mode = 1

Definition at line 48 of file mfe.c.

Referenced by receive_trigger_event(), scheduler(), and send_event().

5.25.3.30 INT run_number

Definition at line 57 of file mfe.c.

Referenced by close_buffers(), cm_transition(), display(), el_submit(), register_equipment(), scheduler(), tr_pause(), tr_resume(), tr_start(), and tr_stop().

5.25.3.31 INT run_state

Definition at line 56 of file mfe.c.

Referenced by close_buffers(), display(), handFlush(), main(), register_equipment(), scheduler(), tr_pause(), tr_resume(), tr_start(), and tr_stop().

5.25.3.32 BOOL slowcont_eq = FALSE

Definition at line 96 of file mfe.c.

Referenced by register_equipment().

5.25.3.33 volatile int stop_all_threads = 0

Definition at line 102 of file mfe.c.

Referenced by main().

5.26 mhttpd.dox File Reference

5.27 midas.c File Reference

5.27.1 Detailed Description

The main core C-code for Midas.

Definition in file midas.c.

Data Structures

- struct TR_CLIENT

Functions

- INT cm_get_error (INT code, char ∗string)
- INT cm_set_msg_print (INT system_mask, INT user_mask, int(∗func)(const char ∗))
- INT cm_msg_log (INT message_type, const char ∗message)
- INT cm_msg_log1 (INT message_type, const char ∗message, const char ∗facility)
- INT cm_msg (INT message_type, const char ∗filename, INT line, const char ∗routine, const char ∗format,...)
- INT cm_msg1 (INT message_type, const char ∗filename, INT line, const char ∗facility, const char ∗routine, const char ∗format,...)
- INT cm_msg_register (void(∗func)(HNDLE, HNDLE, EVENT_HEADER ∗, void ∗))
- INT cm_msg_retrieve (INT n_message, char ∗message, INT buf_size)
- INT cm_synchronize (DWORD ∗seconds)
- INT cm_asctime (char ∗str, INT buf_size)
- INT cm_time (DWORD ∗t)
- char ∗ cm_get_version ()
- int cm_get_revision ()
- INT cm_set_path (char ∗path)
- INT cm_get_path (char ∗path)
- INT cm_scan_experiments (void)
- INT cm_delete_client_info (HNDLE hDB, INT pid)
- INT cm_check_client (HNDLE hDB, HNDLE hKeyClient)
- INT cm_set_client_info (HNDLE hDB, HNDLE ∗hKeyClient, char ∗host_name, char ∗client_name, INT hw_type, char ∗password, DWORD watchdog_timeout)
- INT cm_get_client_info (char ∗client_name)
- INT cm_get_environment (char ∗host_name, int host_name_size, char ∗exp_name, int exp_name_size)

- INT cm_connect_experiment (char ∗host_name, char ∗exp_name, char ∗client_name, void(∗func)(char ∗))
- INT cm_connect_experiment1 (char ∗host_name, char ∗exp_name, char ∗client_name, void(∗func)(char ∗), INT odb_size, DWORD watchdog_timeout)
- INT cm_list_experiments (char ∗host_name, char exp_name[MAX_EXPERIMENT][NAME_LENGTH])
- INT cm_select_experiment (char ∗host_name, char ∗exp_name)
- INT cm_connect_client (char ∗client_name, HNDLE ∗hConn)
- INT cm_disconnect_client (HNDLE hConn, BOOL bShutdown)
- INT cm_disconnect_experiment (void)
- INT cm_set_experiment_database (HNDLE hDB, HNDLE hKeyClient)
- INT cm_get_experiment_database (HNDLE ∗hDB, HNDLE ∗hKeyClient)
- int bm_validate_client_index (const BUFFER ∗buf)
- INT cm_set_watchdog_params (BOOL call_watchdog, DWORD timeout)
- INT cm_get_watchdog_params (BOOL ∗call_watchdog, DWORD ∗timeout)
- INT cm_get_watchdog_info (HNDLE hDB, char ∗client_name, DWORD ∗timeout, DWORD ∗last)
- INT cm_register_transition (INT transition, INT(∗func)(INT, char ∗), INT sequence_number)
- INT cm_set_transition_sequence (INT transition, INT sequence_number)
- INT cm_register_deferred_transition (INT transition, BOOL(∗func)(INT, BOOL))
- INT cm_check_deferred_transition ()
- INT cm_transition (INT transition, INT run_number, char ∗errstr, INT errstr_size, INT async_flag, INT debug_flag)
- INT cm_yield (INT millisec)
- INT cm_execute (char ∗command, char ∗result, INT bufsize)
- INT bm_match_event (short int event_id, short int trigger_mask, EVENT_HEADER ∗pevent)
- INT bm_open_buffer (char ∗buffer_name, INT buffer_size, INT ∗buffer_handle)
- INT bm_close_buffer (INT buffer_handle)
- INT bm_close_all_buffers (void)
- INT cm_shutdown (char ∗name, BOOL bUnique)
- INT cm_exist (char ∗name, BOOL bUnique)
- INT cm_cleanup (char ∗client_name, BOOL ignore_timeout)
- INT bm_set_cache_size (INT buffer_handle, INT read_size, INT write_size)
- INT bm_compose_event (EVENT_HEADER ∗event_header, short int event_id, short int trigger_mask, DWORD size, DWORD serial)
- INT bm_request_event (HNDLE buffer_handle, short int event_id, short int trigger_mask, INT sampling_type, HNDLE ∗request_id, void(∗func)(HNDLE, HNDLE, EVENT_HEADER ∗, void ∗))
- INT bm_remove_event_request (INT buffer_handle, INT request_id)
- INT bm_delete_request (INT request_id)

- INT bm_send_event (INT buffer_handle, void ∗source, INT buf_size, INT async_flag)
- INT bm_flush_cache (INT buffer_handle, INT async_flag)
- INT bm_receive_event (INT buffer_handle, void ∗destination, INT ∗buf_size, INT async_flag)
- INT bm_skip_event (INT buffer_handle)
- INT bm_push_event (char ∗buffer_name)
- INT bm_check_buffers ()
- INT bm_empty_buffers ()
- INT rpc_register_client (char ∗name, RPC_LIST ∗list)
- INT rpc_register_functions (RPC_LIST ∗new_list, INT(∗func)(INT, void ∗∗))
- INT rpc_set_option (HNDLE hConn, INT item, INT value)
- INT rpc_send_event (INT buffer_handle, void ∗source, INT buf_size, INT async_flag, INT mode)
- INT rpc_flush_event ()
- void bk_init (void ∗event)
- void bk_init32 (void ∗event)
- INT bk_size (void ∗event)
- void bk_create (void ∗event, const char ∗name, WORD type, void ∗pdata)
- INT bk_close (void ∗event, void ∗pdata)
- INT bk_list (void ∗event, char ∗bklist)
- INT bk_locate (void ∗event, const char ∗name, void ∗pdata)
- INT bk_find (BANK_HEADER ∗pbkh, const char ∗name, DWORD ∗bklen, DWORD ∗bktype, void ∗∗pdata)
- INT bk_iterate (void ∗event, BANK ∗∗pbk, void ∗pdata)
- INT bk_swap (void ∗event, BOOL force)
- INT dm_buffer_create (INT size, INT user_max_event_size)
- int rb_set_nonblocking ()
- int rb_create (int size, int max_event_size, int ∗handle)
- int rb_delete (int handle)
- int rb_get_wp (int handle, void ∗∗p, int millisec)
- int rb_increment_wp (int handle, int size)
- int rb_get_rp (int handle, void ∗∗p, int millisec)
- int rb_increment_rp (int handle, int size)
- int rb_get_buffer_level (int handle, int ∗n_bytes)

Variables

- HNDLE _hKeyClient = 0

5.27.2 Variable Documentation

5.27.2.1 char∗ svn_revision = "$Rev: 4024 $"

Definition at line 21 of file midas.c.

Referenced by cm_get_revision().

5.28 midas.dox File Reference

5.29 midas.h File Reference

5.29.1 Detailed Description

The main include file

Definition in file midas.h.

Data Structures

- struct EVENT_HEADER
- struct EVENT_REQUEST
- struct BUFFER_CLIENT
- struct BUFFER_HEADER
- struct BUFFER
- struct KEY
- struct KEYLIST
- struct BUS_DRIVER
- struct DD_MT_CHANNEL
- struct DD_MT_BUFFER
- struct DEVICE_DRIVER
- struct EQUIPMENT_INFO
- struct EQUIPMENT_STATS
- struct eqpmnt
- struct BANK_HEADER
- struct BANK
- struct BANK32
- struct TAG
- struct BANK_LIST
- struct ANA_MODULE
- struct AR_INFO
- struct AR_STATS
- struct ANALYZE_REQUEST

- struct ANA_OUTPUT_INFO
- struct ANA_TEST
- struct HIST_RECORD
- struct DEF_RECORD
- struct INDEX_RECORD
- struct HISTORY
- struct RUNINFO
- struct PROGRAM_INFO
- struct ALARM_CLASS
- struct ALARM

Defines

- #define MAX_EVENT_SIZE 0x400000
- #define TAPE_BUFFER_SIZE 0x8000
- #define NET_TCP_SIZE 0xFFFF
- #define OPT_TCP_SIZE 8192
- #define NET_UDP_SIZE 8192
- #define EVENT_BUFFER_NAME "SYSTEM"
- #define DEFAULT_ODB_SIZE 0x100000
- #define NAME_LENGTH 32
- #define HOST_NAME_LENGTH 256
- #define MAX_CLIENTS 64
- #define MAX_EVENT_REQUESTS 10
- #define MAX_OPEN_RECORDS 256
- #define MAX_ODB_PATH 256
- #define MAX_EXPERIMENT 32
- #define BANKLIST_MAX 64
- #define STRING_BANKLIST_MAX BANKLIST_MAX ∗ 4
- #define DEFAULT_RPC_TIMEOUT 10000
- #define DEFAULT_WATCHDOG_TIMEOUT 10000
- #define STATE_STOPPED 1
- #define STATE_PAUSED 2
- #define STATE_RUNNING 3
- #define FORMAT_MIDAS 1
- #define FORMAT_YBOS 2
- #define FORMAT_ASCII 3
- #define FORMAT_FIXED 4
- #define FORMAT_DUMP 5
- #define FORMAT_HBOOK 6
- #define FORMAT_ROOT 7
- #define GET_ALL (1<<0)

- #define GET_SOME (1<<1)
- #define GET_FARM (1<<2)
- #define TID_BYTE 1
- #define TID_SBYTE 2
- #define TID_CHAR 3
- #define TID_WORD 4
- #define TID_SHORT 5
- #define TID_DWORD 6
- #define TID_INT 7
- #define TID_BOOL 8
- #define TID_FLOAT 9
- #define TID_DOUBLE 10
- #define TID_BITFIELD 11
- #define TID_STRING 12
- #define TID_ARRAY 13
- #define TID_STRUCT 14
- #define TID_KEY 15
- #define TID_LINK 16
- #define TID_LAST 17
- #define SYNC 0
- #define MODE_READ (1<<0)
- #define RPC_OTIMEOUT 1
- #define WF_WATCH_ME (1<<0)
- #define TR_START (1<<0)
- #define TR_STOP (1<<1)
- #define TR_PAUSE (1<<2)
- #define TR_RESUME (1<<3)
- #define EQ_PERIODIC (1<<0)
- #define EQ_POLLED (1<<1)
- #define EQ_INTERRUPT (1<<2)
- #define EQ_MULTITHREAD (1<<3)
- #define EQ_SLOW (1<<4)
- #define EQ_MANUAL_TRIG (1<<5)
- #define EQ_FRAGMENTED (1<<6)
- #define EQ_EB (1<<7)
- #define RO_RUNNING (1<<0)
- #define RO_STOPPED (1<<1)
- #define RO_PAUSED (1<<2)
- #define RO_BOR (1<<3)
- #define RO_EOR (1<<4)
- #define RO_PAUSE (1<<5)
- #define RO_RESUME (1<<6)
- #define RO_TRANSITIONS (RO_BOR|RO_EOR|RO_PAUSE|RO_RESUME)

- #define RO_ALWAYS (0xFF)
- #define RO_ODB (1<<8)
- #define CH_BS 8
- #define LAM_SOURCE(c, s) (c<<24 | ((s) & 0xFFFFFF))
- #define LAM_STATION(s) (1<<(s-1))
- #define LAM_SOURCE_CRATE(c) (c>>24)
- #define LAM_SOURCE_STATION(s) ((s) & 0xFFFFFF)
- #define CNAF 0x1
- #define MAX(a, b) (((a) > (b)) ? (a) : (b))
- #define MIN(a, b) (((a) < (b)) ? (a) : (b))
- #define ALIGN8(x) (((x)+7) & ∼7)
- #define VALIGN(adr, align) (((POINTER_T) (adr)+align-1) & ∼(align-1))
- #define MT_ERROR (1<<0)
- #define MT_INFO (1<<1)
- #define MT_DEBUG (1<<2)
- #define MT_USER (1<<3)
- #define MT_LOG (1<<4)
- #define MT_TALK (1<<5)
- #define MT_CALL (1<<6)
- #define MT_ALL 0xFF
- #define MERROR MT_ERROR, __FILE__, __LINE__
- #define MINFO MT_INFO, __FILE__, __LINE__
- #define MDEBUG MT_DEBUG, __FILE__, __LINE__
- #define MUSER MT_USER, __FILE__, __LINE__
- #define MLOG MT_LOG, __FILE__, __LINE__
- #define MTALK MT_TALK, __FILE__, __LINE__
- #define MCALL MT_CALL, __FILE__, __LINE__
- #define SUCCESS 1
- #define CM_SUCCESS 1
- #define CM_SET_ERROR 102
- #define CM_NO_CLIENT 103
- #define CM_DB_ERROR 104
- #define CM_UNDEF_EXP 105
- #define CM_VERSION_MISMATCH 106
- #define CM_SHUTDOWN 107
- #define CM_WRONG_PASSWORD 108
- #define CM_UNDEF_ENVIRON 109
- #define CM_DEFERRED_TRANSITION 110
- #define CM_TRANSITION_IN_PROGRESS 111
- #define CM_TIMEOUT 112
- #define CM_INVALID_TRANSITION 113
- #define CM_TOO_MANY_REQUESTS 114
- #define BM_SUCCESS 1

- #define BM_CREATED 202
- #define BM_NO_MEMORY 203
- #define BM_INVALID_NAME 204
- #define BM_INVALID_HANDLE 205
- #define BM_NO_SLOT 206
- #define BM_NO_MUTEX 207
- #define BM_NOT_FOUND 208
- #define BM_ASYNC_RETURN 209
- #define BM_TRUNCATED 210
- #define BM_MULTIPLE_HOSTS 211
- #define BM_MEMSIZE_MISMATCH 212
- #define BM_CONFLICT 213
- #define BM_EXIT 214
- #define BM_INVALID_PARAM 215
- #define BM_MORE_EVENTS 216
- #define BM_INVALID_MIXING 217
- #define BM_NO_SHM 218
- #define DB_SUCCESS 1
- #define DB_CREATED 302
- #define DB_NO_MEMORY 303
- #define DB_INVALID_NAME 304
- #define DB_INVALID_HANDLE 305
- #define DB_NO_SLOT 306
- #define DB_NO_MUTEX 307
- #define DB_MEMSIZE_MISMATCH 308
- #define DB_INVALID_PARAM 309
- #define DB_FULL 310
- #define DB_KEY_EXIST 311
- #define DB_NO_KEY 312
- #define DB_KEY_CREATED 313
- #define DB_TRUNCATED 314
- #define DB_TYPE_MISMATCH 315
- #define DB_NO_MORE_SUBKEYS 316
- #define DB_FILE_ERROR 317
- #define DB_NO_ACCESS 318
- #define DB_STRUCT_SIZE_MISMATCH 319
- #define DB_OPEN_RECORD 320
- #define DB_OUT_OF_RANGE 321
- #define DB_INVALID_LINK 322
- #define DB_CORRUPTED 323
- #define DB_STRUCT_MISMATCH 324
- #define DB_TIMEOUT 325
- #define DB_VERSION_MISMATCH 326

- #define SS_SUCCESS 1
- #define SS_CREATED 402
- #define SS_NO_MEMORY 403
- #define SS_INVALID_NAME 404
- #define SS_INVALID_HANDLE 405
- #define SS_INVALID_ADDRESS 406
- #define SS_FILE_ERROR 407
- #define SS_NO_MUTEX 408
- #define SS_NO_PROCESS 409
- #define SS_NO_THREAD 410
- #define SS_SOCKET_ERROR 411
- #define SS_TIMEOUT 412
- #define SS_SERVER_RECV 413
- #define SS_CLIENT_RECV 414
- #define SS_ABORT 415
- #define SS_EXIT 416
- #define SS_NO_TAPE 417
- #define SS_DEV_BUSY 418
- #define SS_IO_ERROR 419
- #define SS_TAPE_ERROR 420
- #define SS_NO_DRIVER 421
- #define SS_END_OF_TAPE 422
- #define SS_END_OF_FILE 423
- #define SS_FILE_EXISTS 424
- #define SS_NO_SPACE 425
- #define SS_INVALID_FORMAT 426
- #define SS_NO_ROOT 427
- #define SS_SIZE_MISMATCH 428
- #define RPC_SUCCESS 1
- #define RPC_ABORT SS_ABORT
- #define RPC_NO_CONNECTION 502
- #define RPC_NET_ERROR 503
- #define RPC_TIMEOUT 504
- #define RPC_EXCEED_BUFFER 505
- #define RPC_NOT_REGISTERED 506
- #define RPC_CONNCLOSED 507
- #define RPC_INVALID_ID 508
- #define RPC_SHUTDOWN 509
- #define RPC_NO_MEMORY 510
- #define RPC_DOUBLE_DEFINED 511
- #define RPC_MUTEX_TIMEOUT 512
- #define FE_SUCCESS 1
- #define FE_ERR_ODB 602

- #define FE_ERR_HW 603
- #define FE_ERR_DISABLED 604
- #define FE_ERR_DRIVER 605
- #define HS_SUCCESS 1
- #define HS_FILE_ERROR 702
- #define HS_NO_MEMORY 703
- #define HS_TRUNCATED 704
- #define HS_WRONG_INDEX 705
- #define HS_UNDEFINED_EVENT 706
- #define HS_UNDEFINED_VAR 707
- #define FTP_SUCCESS 1
- #define FTP_NET_ERROR 802
- #define FTP_FILE_ERROR 803
- #define FTP_RESPONSE_ERROR 804
- #define FTP_INVALID_ARG 805
- #define EL_SUCCESS 1
- #define EL_FILE_ERROR 902
- #define EL_NO_MESSAGE 903
- #define EL_TRUNCATED 904
- #define EL_FIRST_MSG 905
- #define EL_LAST_MSG 906
- #define AL_SUCCESS 1
- #define AL_INVALID_NAME 1002
- #define AL_ERROR_ODB 1003
- #define AL_RESET 1004
- #define CMD_INIT 1
- #define CMD_WRITE 100
- #define CMD_INTERRUPT_ENABLE 100
- #define BD_GETS(s, z, p, t) info → bd(CMD_GETS, info → bd_info, s, z, p, t)
- #define ANA_CONTINUE 1
- #define TRIGGER_MASK(e) ((((EVENT_HEADER ∗) e)-1) → trigger_mask)
- #define EVENT_ID(e) ((((EVENT_HEADER ∗) e)-1) → event_id)
- #define SERIAL_NUMBER(e) ((((EVENT_HEADER ∗) e)-1) → serial_number)
- #define TIME_STAMP(e) ((((EVENT_HEADER ∗) e)-1) → time_stamp)
- #define EVENTID_BOR ((short int) 0x8000)
- #define EVENTID_EOR ((short int) 0x8001)
- #define EVENTID_MESSAGE ((short int) 0x8002)
- #define EVENTID_FRAG1 ((unsigned short) 0xC000)
- #define MIDAS_MAGIC 0x494d
- #define DF_INPUT (1<<0)
- #define DF_OUTPUT (1<<1)
- #define DF_PRIO_DEVICE (1<<2)

- #define DF_READ_ONLY (1<<3)
- #define BANK_FORMAT_VERSION 1
- #define BANK_FORMAT_32BIT (1<<4)
- #define AT_INTERNAL 1
- #define AT_PROGRAM 2
- #define AT_EVALUATED 3
- #define AT_PERIODIC 4
- #define AT_LAST 4

5.30 mrpc.c File Reference

5.30.1 Detailed Description

The Midas RPC file

Definition in file mrpc.c.

Variables

- RPC_LIST rpc_list_library [ ]
- RPC_LIST rpc_list_system [ ]

5.31 mrpc.h File Reference

5.31.1 Detailed Description

The mrpc include file

Definition in file mrpc.h.

Defines

- #define RPC_CM_SET_CLIENT_INFO 11000
- #define RPC_CM_SET_WATCHDOG_PARAMS 11001
- #define RPC_CM_CLEANUP 11002
- #define RPC_CM_GET_WATCHDOG_INFO 11003
- #define RPC_CM_MSG_LOG 11004
- #define RPC_CM_EXECUTE 11005

- #define RPC_CM_SYNCHRONIZE 11006
- #define RPC_CM_ASCTIME 11007
- #define RPC_CM_TIME 11008
- #define RPC_CM_MSG 11009
- #define RPC_CM_EXIST 11011
- #define RPC_CM_MSG_RETRIEVE 11012
- #define RPC_CM_MSG_LOG1 11013
- #define RPC_BM_OPEN_BUFFER 11100
- #define RPC_BM_CLOSE_BUFFER 11101
- #define RPC_BM_CLOSE_ALL_BUFFERS 11102
- #define RPC_BM_GET_BUFFER_INFO 11103
- #define RPC_BM_GET_BUFFER_LEVEL 11104
- #define RPC_BM_INIT_BUFFER_COUNTERS 11105
- #define RPC_BM_SET_CACHE_SIZE 11106
- #define RPC_BM_ADD_EVENT_REQUEST 11107
- #define RPC_BM_REMOVE_EVENT_REQUEST 11108
- #define RPC_BM_SEND_EVENT 11109
- #define RPC_BM_FLUSH_CACHE 11110
- #define RPC_BM_RECEIVE_EVENT 11111
- #define RPC_BM_MARK_READ_WAITING 11112
- #define RPC_BM_EMPTY_BUFFERS 11113
- #define RPC_BM_SKIP_EVENT 11114
- #define RPC_DB_OPEN_DATABASE 11200
- #define RPC_DB_CLOSE_DATABASE 11201
- #define RPC_DB_CLOSE_ALL_DATABASES 11202
- #define RPC_DB_CREATE_KEY 11203
- #define RPC_DB_CREATE_LINK 11204
- #define RPC_DB_SET_VALUE 11205
- #define RPC_DB_GET_VALUE 11206
- #define RPC_DB_FIND_KEY 11207
- #define RPC_DB_FIND_LINK 11208
- #define RPC_DB_GET_PATH 11209
- #define RPC_DB_DELETE_KEY 11210
- #define RPC_DB_ENUM_KEY 11211
- #define RPC_DB_GET_KEY 11212
- #define RPC_DB_GET_DATA 11213
- #define RPC_DB_SET_DATA 11214
- #define RPC_DB_SET_DATA_INDEX 11215
- #define RPC_DB_SET_MODE 11216
- #define RPC_DB_GET_RECORD_SIZE 11219
- #define RPC_DB_GET_RECORD 11220
- #define RPC_DB_SET_RECORD 11221
- #define RPC_DB_ADD_OPEN_RECORD 11222

- #define RPC_DB_REMOVE_OPEN_RECORD 11223
- #define RPC_DB_SAVE 11224
- #define RPC_DB_LOAD 11225
- #define RPC_DB_SET_CLIENT_NAME 11226
- #define RPC_DB_RENAME_KEY 11227
- #define RPC_DB_ENUM_LINK 11228
- #define RPC_DB_REORDER_KEY 11229
- #define RPC_DB_CREATE_RECORD 11230
- #define RPC_DB_GET_DATA_INDEX 11231
- #define RPC_DB_GET_KEY_TIME 11232
- #define RPC_DB_GET_OPEN_RECORDS 11233
- #define RPC_DB_FLUSH_DATABASE 11235
- #define RPC_DB_SET_DATA_INDEX2 11236
- #define RPC_DB_GET_KEY_INFO 11237
- #define RPC_DB_GET_DATA1 11238
- #define RPC_DB_SET_NUM_VALUES 11239
- #define RPC_DB_CHECK_RECORD 11240
- #define RPC_DB_GET_NEXT_LINK 11241
- #define RPC_HS_SET_PATH 11300
- #define RPC_HS_DEFINE_EVENT 11301
- #define RPC_HS_WRITE_EVENT 11302
- #define RPC_HS_COUNT_EVENTS 11303
- #define RPC_HS_ENUM_EVENTS 11304
- #define RPC_HS_COUNT_VARS 11305
- #define RPC_HS_ENUM_VARS 11306
- #define RPC_HS_READ 11307
- #define RPC_HS_GET_VAR 11308
- #define RPC_HS_GET_EVENT_ID 11309
- #define RPC_EL_SUBMIT 11400
- #define RPC_AL_CHECK 11500
- #define RPC_AL_TRIGGER_ALARM 11501
- #define RPC_RC_TRANSITION 12000
- #define RPC_ANA_CLEAR_HISTOS 13000
- #define RPC_LOG_REWIND 14000
- #define RPC_TEST 15000
- #define RPC_CNAF16 16000
- #define RPC_CNAF24 16001
- #define RPC_MANUAL_TRIG 17000
- #define RPC_ID_WATCHDOG 99997
- #define RPC_ID_SHUTDOWN 99998
- #define RPC_ID_EXIT 99999

5.32 msystem.h File Reference

5.32.1 Detailed Description

The Midas System include file

Definition in file msystem.h.

Data Structures

- struct FREE_DESCRIP
- struct OPEN_RECORD
- struct DATABASE_CLIENT
- struct DATABASE_HEADER
- struct DATABASE
- struct RECORD_LIST
- struct REQUEST_LIST

Defines

- #define DRI_16 (1<<0)
- #define DRI_32 (1<<1)
- #define DRI_64 (1<<2)
- #define DRI_LITTLE_ENDIAN (1<<3)
- #define DRI_BIG_ENDIAN (1<<4)
- #define DRF_IEEE (1<<5)
- #define DRF_G_FLOAT (1<<6)
- #define DR_ASCII (1<<7)
- #define WORD_SWAP(x)
- #define DWORD_SWAP(x)
- #define QWORD_SWAP(x)

5.33 mvmestd.h File Reference

5.33.1 Detailed Description

The Midas VME include file

Definition in file mvmestd.h.

Data Structures

- struct MVME_INTERFACE

Defines

- #define MVME_SUCCESS 1
- #define MVME_DMODE_D8 1
- #define MVME_DMODE_D16 2
- #define MVME_DMODE_D32 3
- #define MVME_DMODE_D64 4
- #define MVME_DMODE_RAMD16 5
- #define MVME_DMODE_RAMD32 6
- #define MVME_DMODE_LM 7
- #define MVME_BLT_NONE 1
- #define MVME_BLT_BLT32 2
- #define MVME_BLT_MBLT64 3
- #define MVME_BLT_2EVME 4
- #define MVME_BLT_2ESST 5
- #define MVME_BLT_BLT32FIFO 6
- #define MVME_BLT_MBLT64FIFO 7
- #define MVME_BLT_2EVMEFIFO 8
- #define MVME_AM_A32_SB (0x0F)
- #define MVME_AM_A32_SP (0x0E)
- #define MVME_AM_A32_SD (0x0D)
- #define MVME_AM_A32_NB (0x0B)
- #define MVME_AM_A32_NP (0x0A)
- #define MVME_AM_A32_ND (0x09)
- #define MVME_AM_A32_SMBLT (0x0C)
- #define MVME_AM_A32_NMBLT (0x08)
- #define MVME_AM_A24_SB (0x3F)
- #define MVME_AM_A24_SP (0x3E)
- #define MVME_AM_A24_SD (0x3D)
- #define MVME_AM_A24_NB (0x3B)
- #define MVME_AM_A24_NP (0x3A)
- #define MVME_AM_A24_ND (0x39)
- #define MVME_AM_A24_SMBLT (0x3C)
- #define MVME_AM_A24_NMBLT (0x38)
- #define MVME_AM_A16_SD (0x2D)
- #define MVME_AM_A16_ND (0x29)

Functions

- int EXPRT mvme_open (MVME_INTERFACE ∗∗vme, int idx)
- int EXPRT mvme_close (MVME_INTERFACE ∗vme)
- int EXPRT mvme_sysreset (MVME_INTERFACE ∗vme)
- int EXPRT mvme_read (MVME_INTERFACE ∗vme, void ∗dst, mvme_addr_t vme_addr, mvme_size_t n_bytes)
- unsigned int EXPRT mvme_read_value (MVME_INTERFACE ∗vme, mvme_addr_t vme_addr)
- int EXPRT mvme_write (MVME_INTERFACE ∗vme, mvme_addr_t vme_addr, void ∗src, mvme_size_t n_bytes)
- int EXPRT mvme_write_value (MVME_INTERFACE ∗vme, mvme_addr_t vme_addr, unsigned int value)
- int EXPRT mvme_set_am (MVME_INTERFACE ∗vme, int am)
- int EXPRT mvme_get_am (MVME_INTERFACE ∗vme, int ∗am)
- int EXPRT mvme_set_dmode (MVME_INTERFACE ∗vme, int dmode)
- int EXPRT mvme_get_dmode (MVME_INTERFACE ∗vme, int ∗dmode)
- int EXPRT mvme_set_blt (MVME_INTERFACE ∗vme, int mode)
- int EXPRT mvme_get_blt (MVME_INTERFACE ∗vme, int ∗mode)

5.34 myexpt.html File Reference

5.34.1 Variable Documentation

5.34.1.1 <html><head><title> MyExperiment Demo Status</title><metahttp-equiv="Refresh"content="30"></head><body><formname="form1"method="Get Expt&"><tableborder=3cellpadding=2><tr><thbgcolor="#A0A0FF"> Demo Experiment<th bgcolor="#A0A0FF"> Custom Monitor Control</tr><tr><td><b><fontcolor="#ff0000"> Actions

Definition at line 29 of file myexpt.html.

5.34.1.2 scale = 12h&amp

Definition at line 32 of file myexpt.html.

5.34.1.3 width

Initial value:

```
250">
        </th>
        <th> <img src="http:
                        exp=default&amp
```

Definition at line 29 of file myexpt.html.

5.35 newdocfeatures.dox File Reference

5.36 odb.c File Reference

5.36.1 Detailed Description

The Online Database file

Definition in file odb.c.

Functions

- INT db_open_database (char ∗database_name, INT database_size, HNDLE ∗hDB, char ∗client_name)
- INT db_close_database (HNDLE hDB)
- INT db_lock_database (HNDLE hDB)
- INT db_unlock_database (HNDLE hDB)
- INT db_protect_database (HNDLE hDB)
- INT db_create_key (HNDLE hDB, HNDLE hKey, char ∗key_name, DWORD type)
- INT db_create_link (HNDLE hDB, HNDLE hKey, char ∗link_name, char ∗destination)
- INT db_delete_key1 (HNDLE hDB, HNDLE hKey, INT level, BOOL follow_links)
- INT db_delete_key (HNDLE hDB, HNDLE hKey, BOOL follow_links)
- INT db_find_key (HNDLE hDB, HNDLE hKey, char ∗key_name, HNDLE ∗subhKey)
- INT db_set_value (HNDLE hDB, HNDLE hKeyRoot, char ∗key_name, void ∗data, INT data_size, INT num_values, DWORD type)
- INT db_set_value_index (HNDLE hDB, HNDLE hKeyRoot, char ∗key_name, void ∗data, INT data_size, INT index, DWORD type, BOOL truncate)

- INT db_get_value (HNDLE hDB, HNDLE hKeyRoot, char ∗key_name, void ∗data, INT ∗buf_size, DWORD type, BOOL create)
- INT db_enum_key (HNDLE hDB, HNDLE hKey, INT idx, HNDLE ∗subkey_handle)
- INT db_get_key (HNDLE hDB, HNDLE hKey, KEY ∗key)
- INT db_get_key_time (HNDLE hDB, HNDLE hKey, DWORD ∗delta)
- INT db_get_key_info (HNDLE hDB, HNDLE hKey, char ∗name, INT name_size, INT ∗type, INT ∗num_values, INT ∗item_size)
- INT db_get_data (HNDLE hDB, HNDLE hKey, void ∗data, INT ∗buf_size, DWORD type)
- INT db_get_data_index (HNDLE hDB, HNDLE hKey, void ∗data, INT ∗buf_size, INT idx, DWORD type)
- INT db_set_data (HNDLE hDB, HNDLE hKey, void ∗data, INT buf_size, INT num_values, DWORD type)
- INT db_set_data_index (HNDLE hDB, HNDLE hKey, void ∗data, INT data_size, INT idx, DWORD type)
- INT db_load (HNDLE hDB, HNDLE hKeyRoot, char ∗filename, BOOL bRemote)
- INT db_copy (HNDLE hDB, HNDLE hKey, char ∗buffer, INT ∗buffer_size, char ∗path)
- INT db_paste (HNDLE hDB, HNDLE hKeyRoot, char ∗buffer)
- INT db_paste_xml (HNDLE hDB, HNDLE hKeyRoot, char ∗buffer)
- INT db_copy_xml (HNDLE hDB, HNDLE hKey, char ∗buffer, INT ∗buffer_size)
- INT db_save (HNDLE hDB, HNDLE hKey, char ∗filename, BOOL bRemote)
- INT db_save_xml (HNDLE hDB, HNDLE hKey, char ∗filename)
- INT db_save_struct (HNDLE hDB, HNDLE hKey, char ∗file_name, char ∗struct_name, BOOL append)
- INT db_sprintf (char ∗string, void ∗data, INT data_size, INT idx, DWORD type)
- INT db_get_record_size (HNDLE hDB, HNDLE hKey, INT align, INT ∗buf_size)
- INT db_get_record (HNDLE hDB, HNDLE hKey, void ∗data, INT ∗buf_size, INT align)
- INT db_set_record (HNDLE hDB, HNDLE hKey, void ∗data, INT buf_size, INT align)
- INT db_create_record (HNDLE hDB, HNDLE hKey, char ∗orig_key_name, char ∗init_str)
- INT db_check_record (HNDLE hDB, HNDLE hKey, char ∗keyname, char ∗rec_str, BOOL correct)
- INT db_open_record (HNDLE hDB, HNDLE hKey, void ∗ptr, INT rec_size, WORD access_mode, void(∗dispatcher)(INT, INT, void ∗), void ∗info)
- INT db_close_record (HNDLE hDB, HNDLE hKey)
- INT db_close_all_records ()
- INT db_update_record (INT hDB, INT hKey, int s)

- INT db_send_changed_records ()

5.37 odbstruct.dox File Reference

5.38 quickstart.dox File Reference

5.39 scaler.c File Reference

5.39.1 Function Documentation

5.39.1.1 INT scaler_accum (EVENT_HEADER ∗, void ∗)

Definition at line 66 of file scaler.c.

5.39.1.2 INT scaler_clear (INT *run_number*)

Definition at line 51 of file scaler.c.

5.39.1.3 INT scaler_eor (INT *run_number*)

Definition at line 59 of file scaler.c.

5.39.2 Variable Documentation

5.39.2.1 double scaler[32]

Definition at line 47 of file scaler.c.

Referenced by scaler_accum(), and scaler_clear().

5.39.2.2 ANA_MODULE scaler_accum_module

Initial value:

```
 {
   "Scaler accumulation",
   "Stefan Ritt",
   scaler_accum,
   scaler_clear,
   scaler_eor,
   NULL,
   NULL,
   NULL,
   0,
   NULL,
}
```

Definition at line 32 of file scaler.c.

5.40 system.c File Reference

5.40.1 Detailed Description

The Midas System file

Definition in file system.c.

Functions

- INT ss_system (char ∗command)
- midas_thread_t ss_thread_create (INT(∗thread_func)(void ∗), void ∗param)
- INT ss_thread_kill (midas_thread_t thread_id)
- DWORD ss_millitime ()
- DWORD ss_time ()
- INT ss_sleep (INT millisec)

5.41 utilities.dox File Reference

5.42 xcustom.odb File Reference

5.43 ybos.c File Reference

5.43.1 Detailed Description

The YBOS file

Definition in file ybos.c.

Functions

- void ybk_init (DWORD ∗plrl)
- void ybk_create (DWORD ∗plrl, char ∗bkname, DWORD bktype, void ∗pbkdat)
- INT ybk_close (DWORD ∗plrl, void ∗pbkdat)
- INT ybk_size (DWORD ∗plrl)
- INT ybk_list (DWORD ∗plrl, char ∗bklist)
- INT ybk_find (DWORD ∗plrl, char ∗bkname, DWORD ∗bklen, DWORD ∗bktype, void ∗∗pbk)
- INT ybk_locate (DWORD ∗plrl, char ∗bkname, void ∗pdata)
- INT ybk_iterate (DWORD ∗plrl, YBOS_BANK_HEADER ∗∗pybkh, void ∗∗pdata)

5.44 ybos.h File Reference

5.44.1 Detailed Description

The YBOS include file

Definition in file ybos.h.

Defines

- #define YBOS_PHYREC_SIZE 8192
- #define YBOS_BUFFER_SIZE 3∗(YBOS_PHYREC_SIZE<<2) + MAX_EVENT_SIZE + 128
- #define YB_BANKLIST_MAX 32
- #define YB_STRING_BANKLIST_MAX YB_BANKLIST_MAX ∗ 4
- #define YB_SUCCESS 1
- #define YB_EVENT_NOT_SWAPPED 2
- #define YB_DONE 2
- #define YB_WRONG_BANK_TYPE -100
- #define YB_BANK_NOT_FOUND -101

- #define YB_SWAP_ERROR -102
- #define YB_NOMORE_SLOT -103
- #define YB_UNKNOWN_FORMAT -104
- #define H_BLOCK_SIZE 0
- #define H_BLOCK_NUM 1
- #define H_HEAD_LEN 2
- #define H_START 3
- #define D_RECORD 1
- #define D_HEADER 2
- #define D_EVTLEN 3
- #define YB_COMPLETE 1
- #define YB_INCOMPLETE 2
- #define YB_NO_RECOVER -1
- #define YB_NO_RUN 0
- #define YB_ADD_RUN 1
- #define DSP_RAW 1
- #define DSP_RAW_SINGLE 2
- #define DSP_BANK 3
- #define DSP_BANK_SINGLE 4
- #define DSP_UNK 0
- #define DSP_DEC 1
- #define DSP_HEX 2
- #define DSP_ASC 3
- #define SWAP_D2WORD(_d2w)
- #define EVID_TRINAT
- #define YBOS_EVID_BANK(__a, __b, __c,__d,__e)
- #define MIDAS_EVID_BANK(__a, __b, __c,__d,__e)
- #define I2_BKTYPE 1
- #define A1_BKTYPE 2
- #define I4_BKTYPE 3
- #define F4_BKTYPE 4
- #define D8_BKTYPE 5
- #define I1_BKTYPE 8
- #define MAX_BKTYPE I1_BKTYPE+1

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:

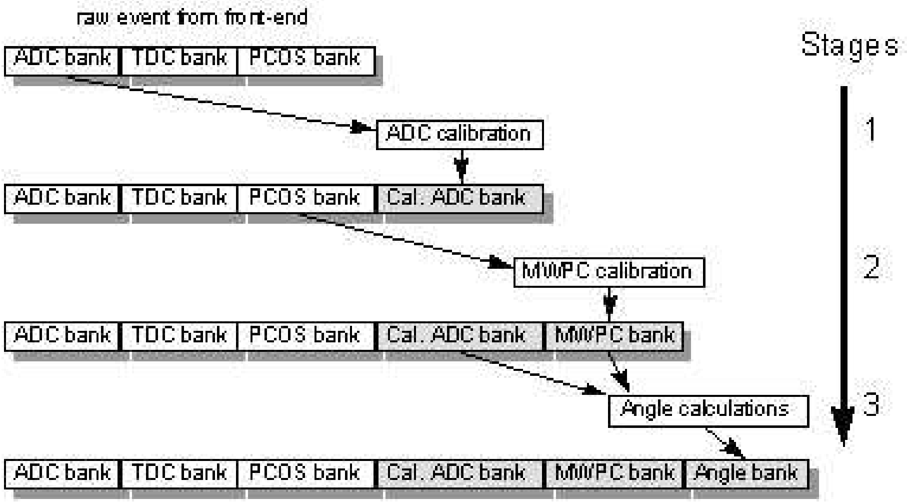


Figure 1: Three stage analyzer.

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
```

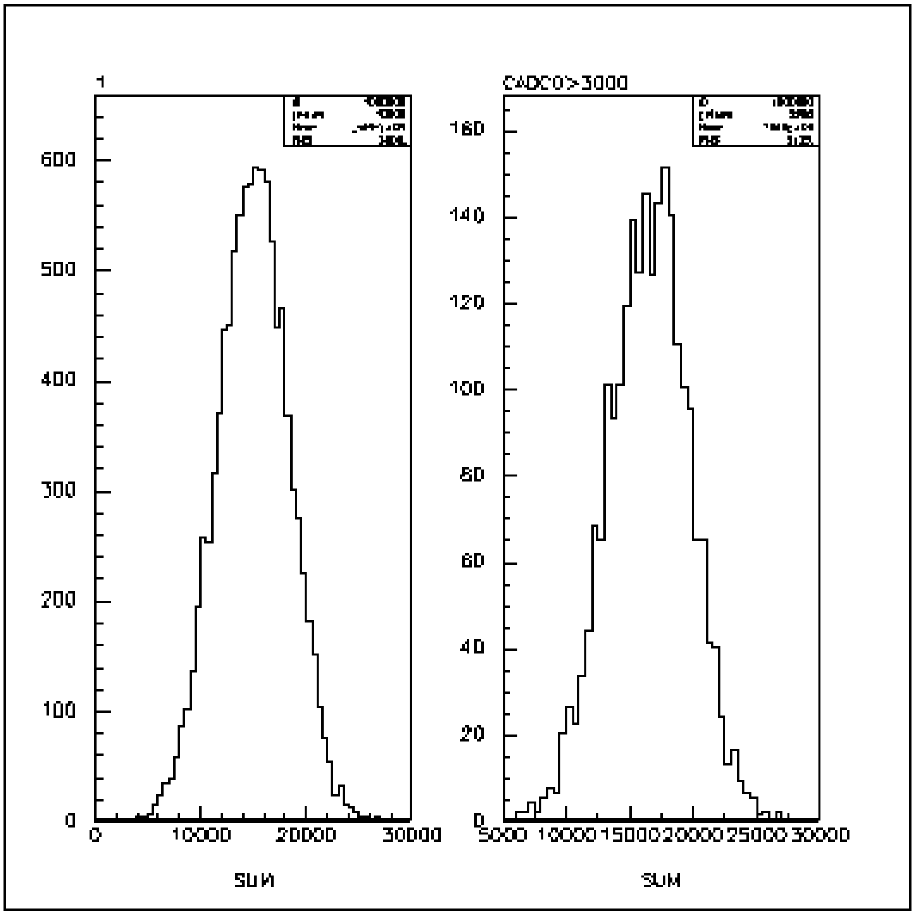


Figure 2: PAW output for online N-tuples.

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 online. 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.

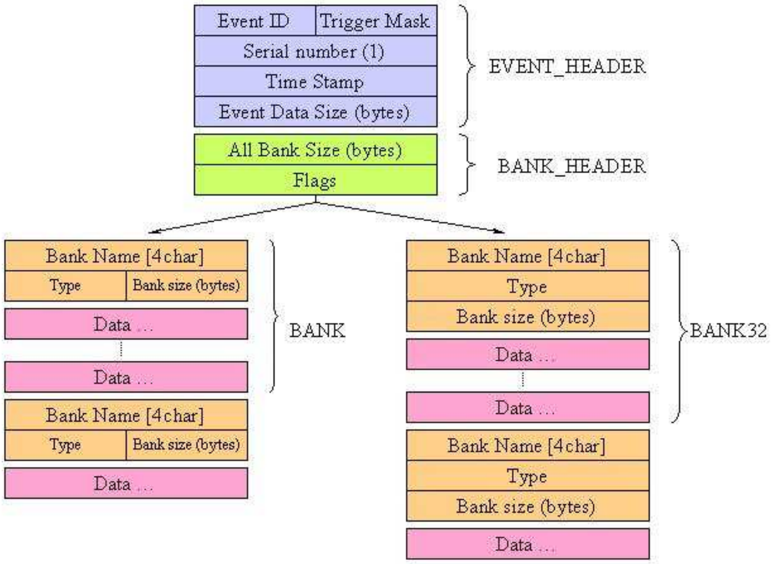


Figure 3: Event and bank headers with data block.

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.

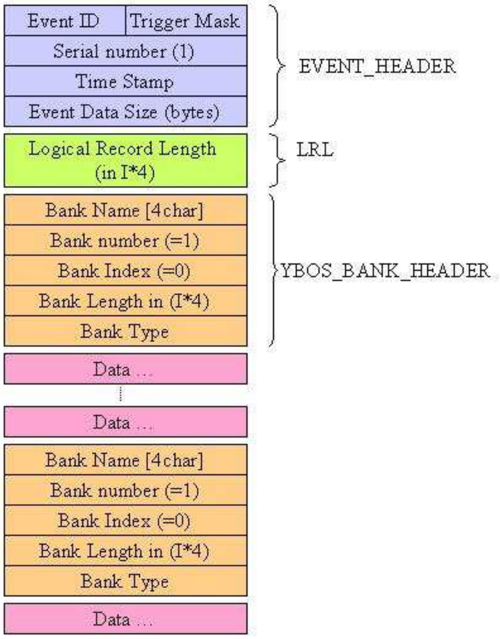


Figure 4: Ybos Event and bank headers with data block.

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.

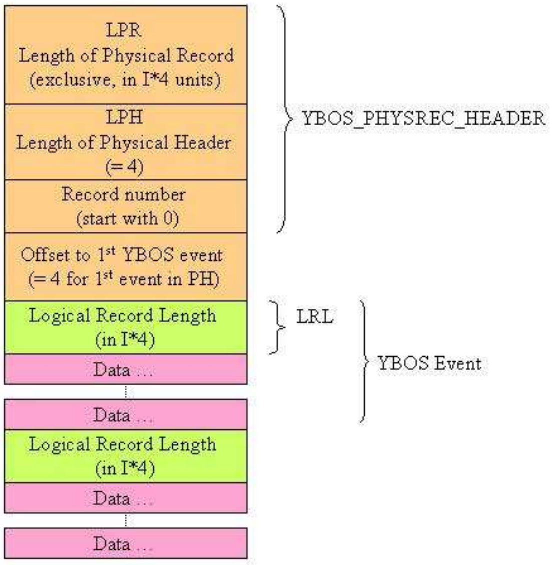


Figure 5: Ybos Physical record structure with data block..

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

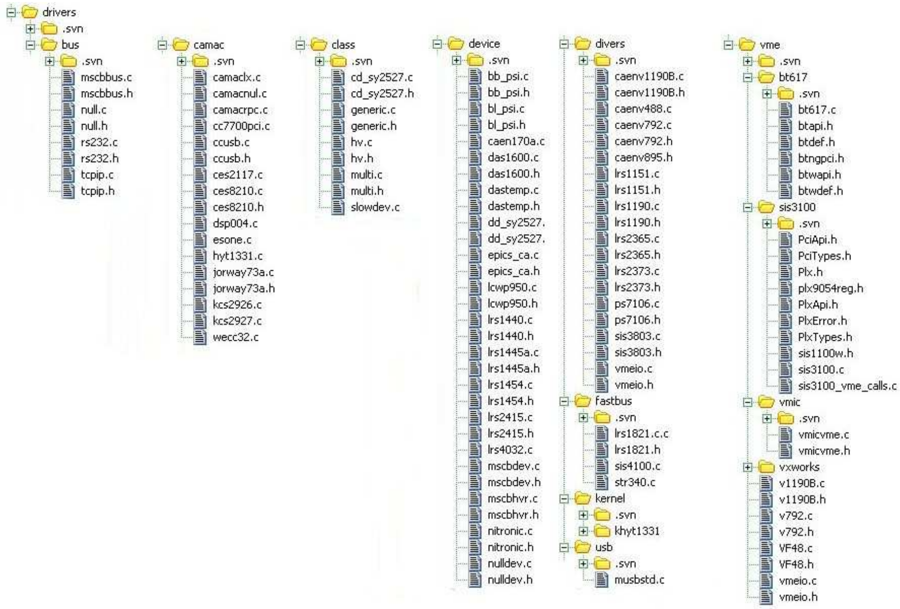


Figure 6: Drivers/ directory structure

- 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 Camac Functions (camxxx) for function description.

6.4.2 ESONE CAMAC standard functions

Not all the functionality of ESONE standard have been fully tested

Please refer to ∗ Camac Functions (Esone) for function description.

6.4.3 Midas VME standard functions

This API provides basic VME access through a **simple** set of functions. Refer to VME Functions (mvme_xxx) 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.

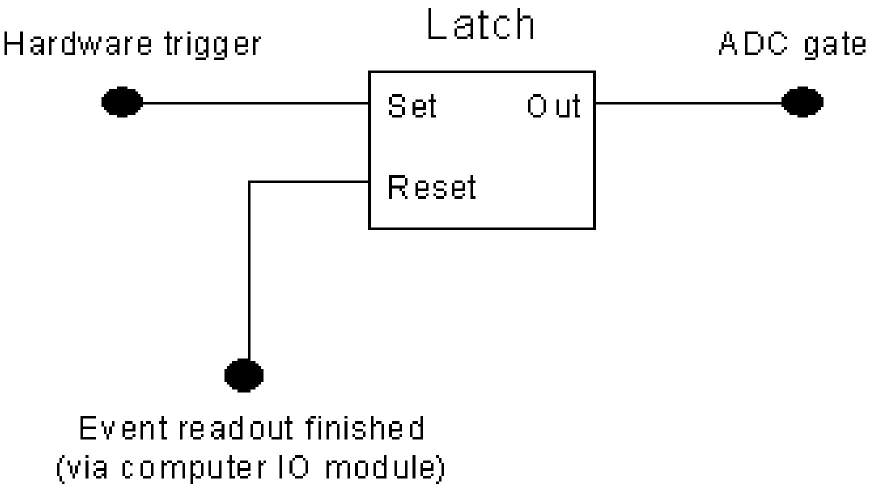


Figure 7: Latched trigger layout.

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

Improved Latched trigger layout.

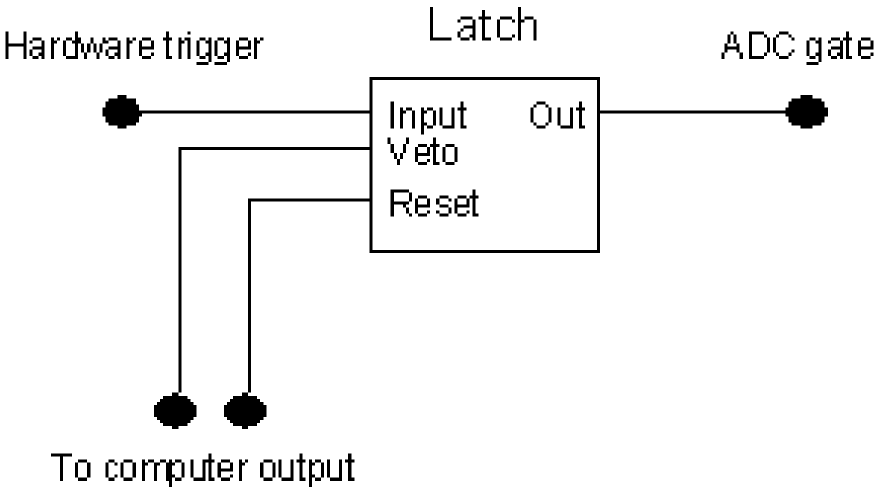


Figure 8: Improved Latched trigger layout.

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.

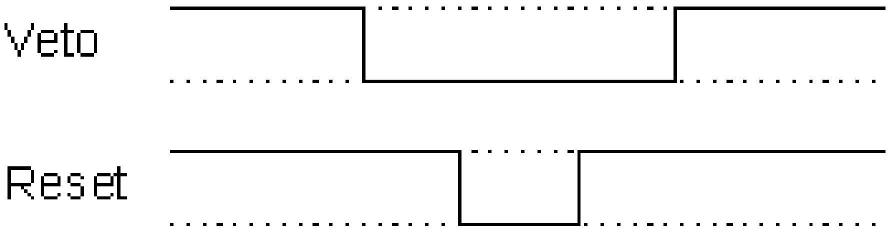


Figure 9: Veto Timing.

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 , MIDAS_MAX_EVENT_SIZE

- 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.

6.5.8 MIDAS_MAX_EVENT_SIZE

By default the Midas package is build with the maximum event size set to 4MB (MAX_EVENT_SIZE/midas.h). This parameter is used for event transfer across network as well, therefore it has to be applied to all the midas client involved in the experiment when different value is required and a complete Midas rebuid needs to be done.

```
> setenv MIDAS_MAX_EVENT_SIZE 8000000
> make
cc -c -g -O3 -Wall -Wuninitialized -Iinclude -Idrivers -I../mxml -Llinux/lib -DINCLUDE_FTPLIB  \
 -DMAX_EVENT_SIZE=800000 -D_LARGEFILE64_SOURCE -DHAVE_MYSQL -I/usr/include/mysql -DHAVE_ROOT -pthrea
-m64 -I/triumfcs/trshare/olchansk/root/root_v5.12.00_SL42_amd64/include -DHAVE_ZLIB -DOS_LINUX -fPIC
-Wno-unused-function -o linux/lib/midas.o src/midas.c
...
```

But at the frontend level, the user can define his own local maximum event size through the **max_event_size** (see frontend examples).

6.5.9 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
```

6.5.10 INCLUDE_FTPLIB

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

6.5.11 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
```

6.5.12 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
```

6.5.13 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).

6.5.14 USE_EVENT_CHANNEL

To be used in conjunction with the DM_DUAL_THREAD.

6.5.15 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).

6.5.16 Environment variables

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

6.5.16.1 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.

6.5.16.2 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.

6.5.16.3 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.

6.5.16.4 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.

6.5.16.5 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.

6.5.16.6 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.

- State Codes & Transition Codes
- Midas Data Types
  - Midas bank examples
- YBOS Bank Types
  - YBOS bank examples

- Midas Code and Libraries

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 pre-composed 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 codes and their corresponding header files.

1. The midas.h & midas.c : Midas abstract layer.
2. The msystem.h & system.c : Midas function implementation.
3. Midas Alarm Functions (al_xxx) : Midas Alarm functions.
4. Midas History Functions (hs_xxx) : Midas History functions.
5. Midas Elog Functions (el_xxx) : Midas Elog functions.
6. The mrpc.h & mrpc.c : Midas RPC functions.
7. The odb.c : Online Database functions.
8. 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
- **rb_xxx**(...) : Ring buffer 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.

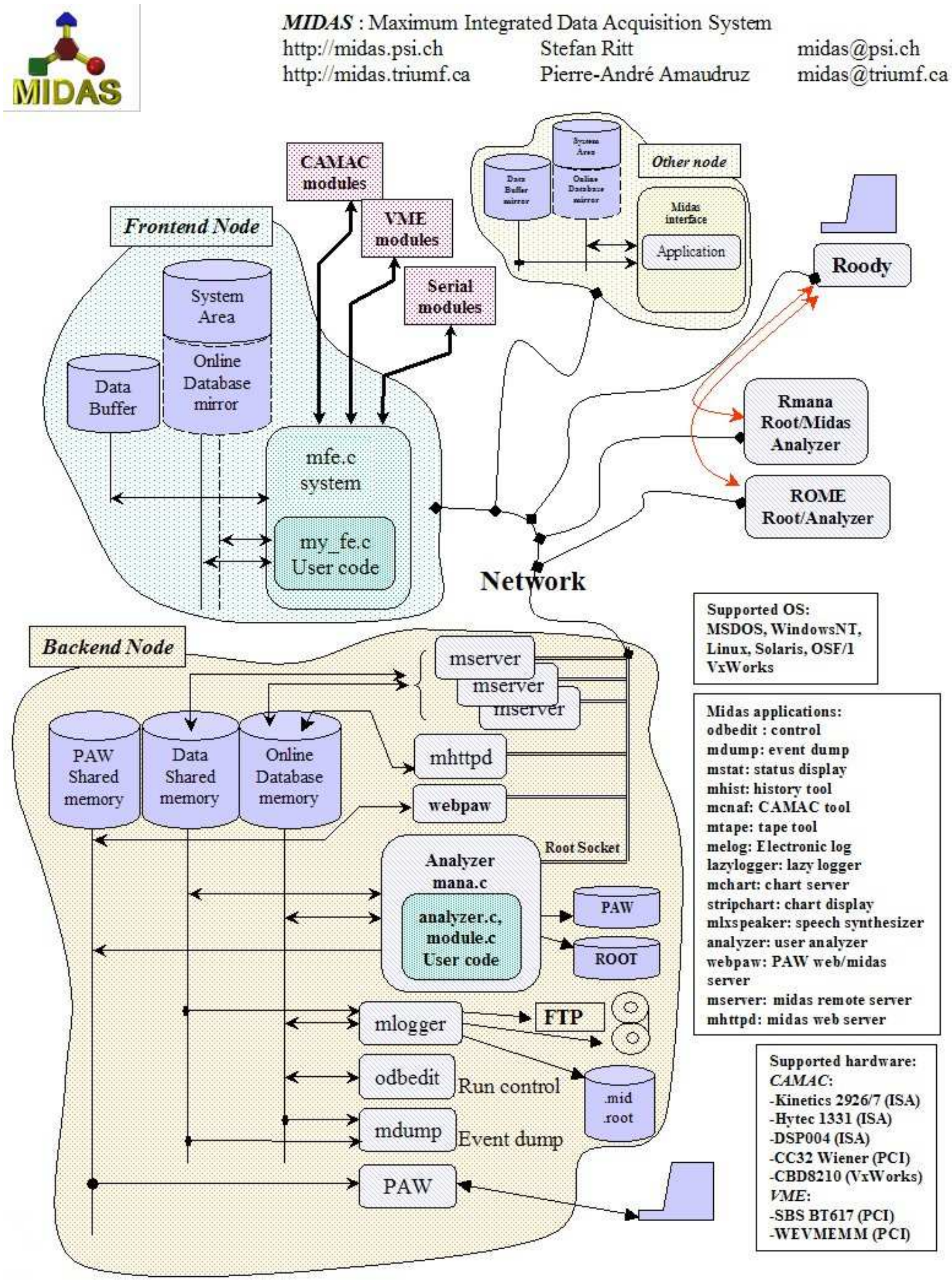


Figure 10: Components

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).

Transitions

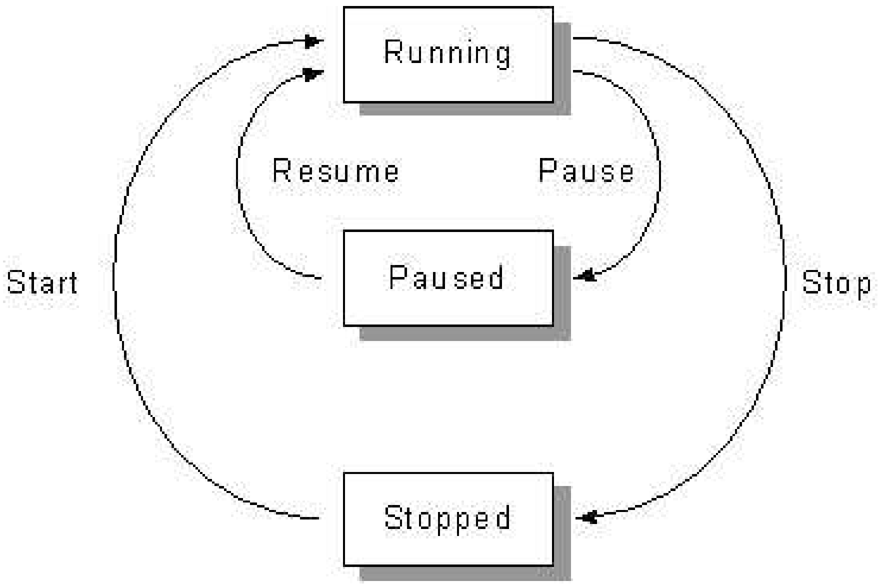


Figure 11: Transitions

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.

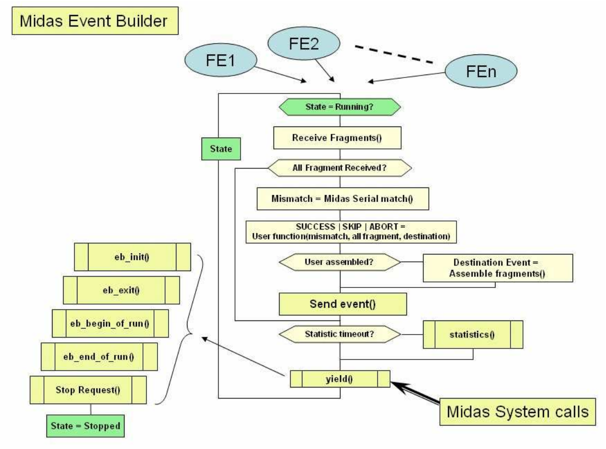


Figure 12: Event Builder Flowchart.

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 Midas 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

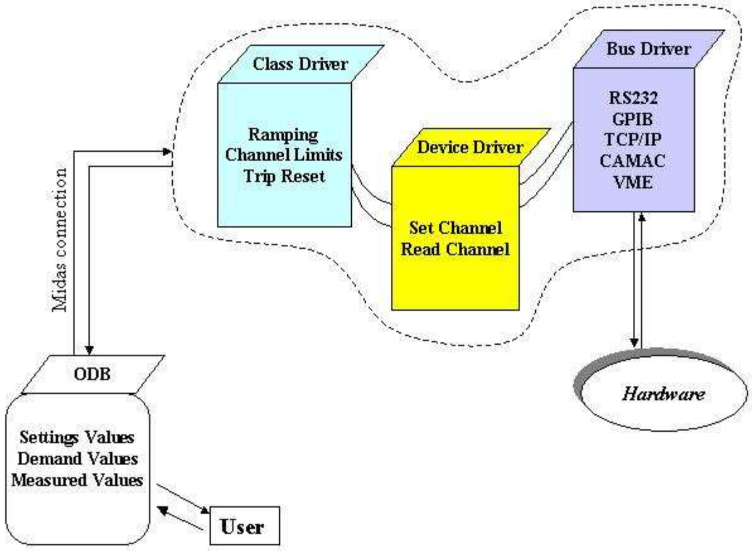


Figure 13: Class driver, Device and Bus driver in the slow control system

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.

Slow Control variables can be later accessed through the web using the midas page (mhttpd task, Equipment page). This can be done setting the variable names under the directory Settings of the corresponding equipment. The variable description is given under History system. The History page will also be automatically activated if the application mlogger task is running.

```
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 MeasureFLOAT   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)
- Slow Control page (Equipment info specific to Slow Control)
- 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
    ```

[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.

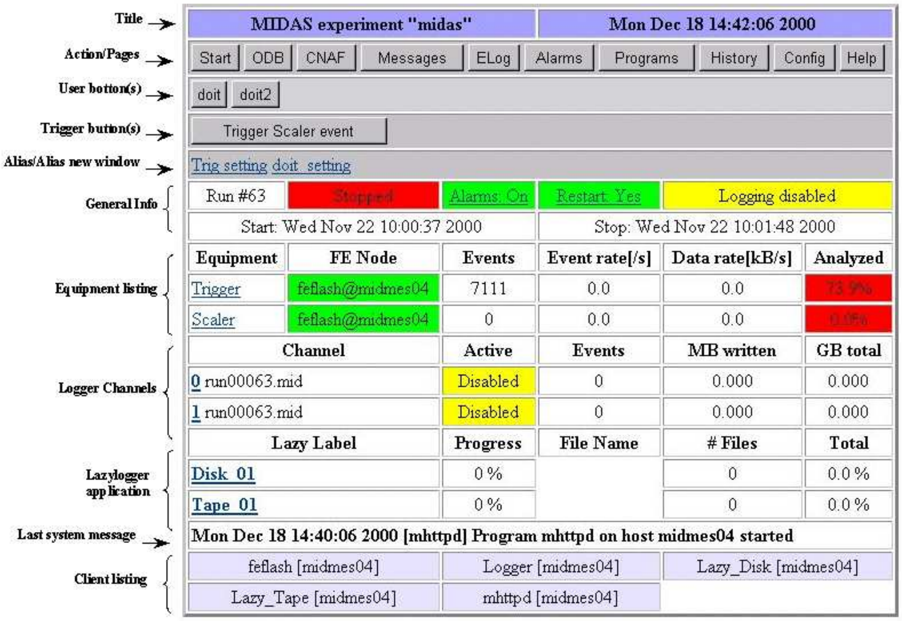
- **Example** : The **Example** will create a shortcut to the defined location in the ODB.

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

- [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



MIDAS experiment "midas" | Mon Dec 18 14:42:06 2000

Trig setting doit setting

Run #63	Stopped	Alarms: On	Restart: Yes	Logging disabled

Start: Wed Nov 22 10:00:37 2000 | Stop: Wed Nov 22 10:01:48 2000

Equipment	FE Node	Events	Event rate[/s]	Data rate[kB/s]	Analyzed
Trigger	feflash@midmes04	7111	0.0	0.0	73.9%
Scaler	feflash@midmes04	0	0.0	0.0	0.0%

Channel	Active	Events	MB written	GB total
0 run00063.mid	Disabled	0	0.000	0.000
1 run00063.mid	Disabled	0	0.000	0.000

Lazy Label	Progress	File Name	# Files	Total
Disk_01	0 %		0	0.0 %
Tape_01	0 %		0	0.0 %

Mon Dec 18 14:40:06 2000 [mhttpd] Program mhttpd on host midmes04 started

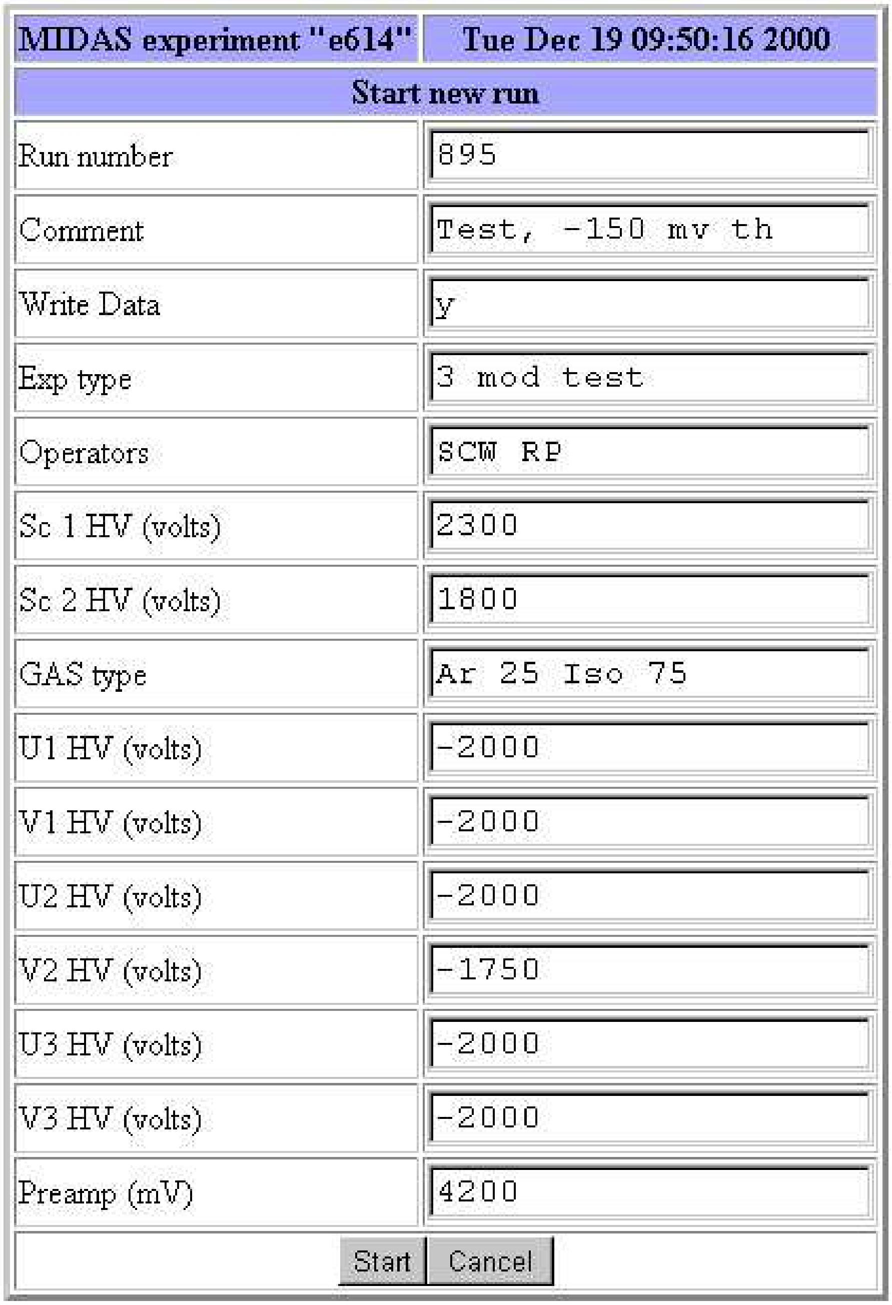
feflash [midmes04]	Logger [midmes04]	Lazy_Disk [midmes04]
Lazy_Tape [midmes04]	mhttpd [midmes04]	

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

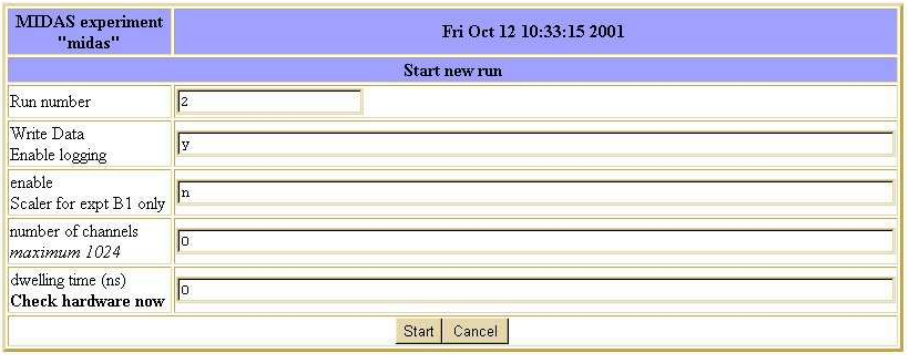
Figure 15: Start run request page.

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.



MIDAS experiment "midas"	Fri Oct 12 10:33:15 2001
Start new run	
Run number	2
Write Data Enable logging	y
enable Scaler for expt B1 only	n
number of channels *maximum 1024*	0
dwelling time (ns) **Check hardware now**	0

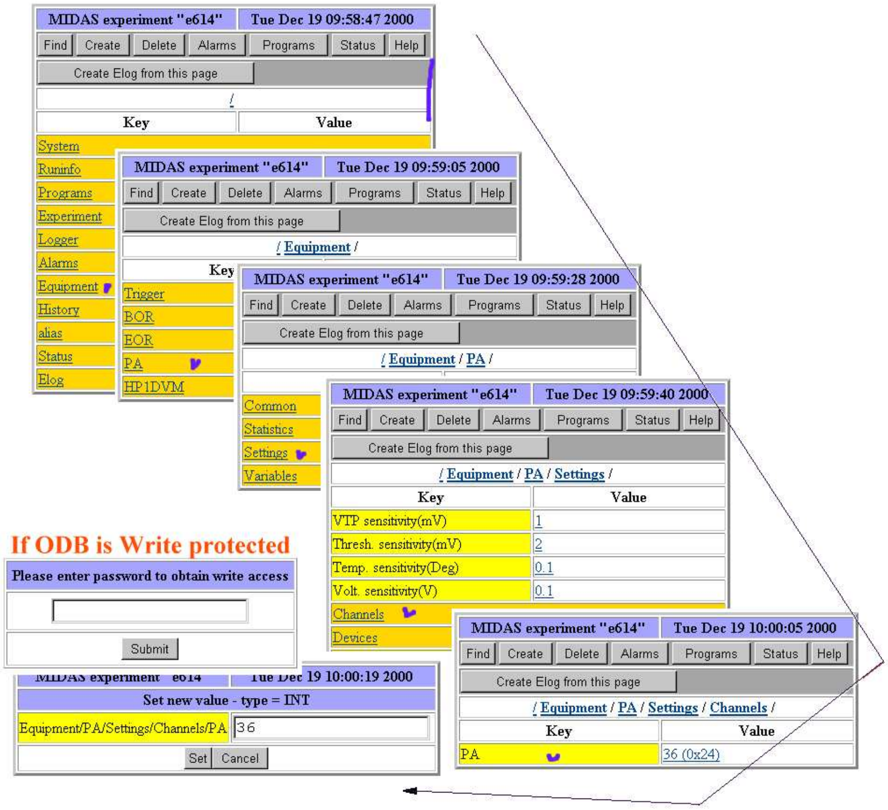
Figure 16: Start run request page.

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.



MIDAS experiment "e614" Tue Dec 19 09:58:47 2000

/

Key	Value
System	
Runinfo	
Programs	
Experiment	
Logger	
Alarms	
Equipment	
History	
alias	
Status	
Elog	

MIDAS experiment "e614" Tue Dec 19 09:59:05 2000

/ Equipment /

Key
Trigger
BOR
EOR
PA
HP1DVM

MIDAS experiment "e614" Tue Dec 19 09:59:28 2000

/ Equipment / PA /

Common
Statistics
Settings
Variables

MIDAS experiment "e614" Tue Dec 19 09:59:40 2000

/ Equipment / PA / Settings /

Key	Value
VTP sensitivity(mV)	1
Thresh. sensitivity(mV)	2
Temp. sensitivity(Deg)	0.1
Volt. sensitivity(V)	0.1
Channels	
Devices	

MIDAS experiment "e614" Tue Dec 19 10:00:05 2000

/ Equipment / PA / Settings / Channels /

Key	Value
PA	36 (0x24)

If ODB is Write protected

Please enter password to obtain write access

MIDAS experiment "e614" Tue Dec 19 10:00:19 2000

Set new value - type = INT

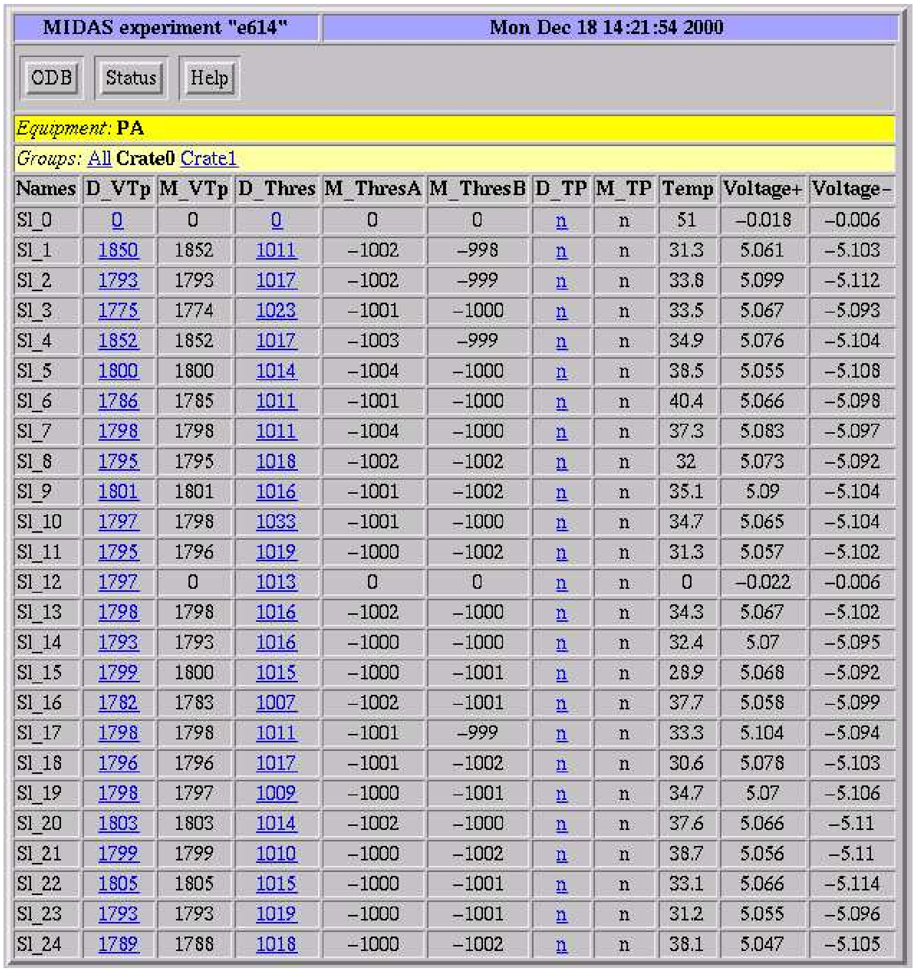
Equipment/PA/Settings/Channels/PA: 36

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 Slow Control page and History system).

Slow control page.



Equipment: **PA**

Groups: All **Crate0** Crate1

Names	D_VTp	M_VTp	D_Thres	M_ThresA	M_ThresB	D_TP	M_TP	Temp	Voltage+	Voltage-
Sl_0	0	0	0	0	0	n	n	51	-0.018	-0.006
Sl_1	1850	1852	1011	-1002	-998	n	n	31.3	5.061	-5.103
Sl_2	1793	1793	1017	-1002	-999	n	n	33.8	5.099	-5.112
Sl_3	1775	1774	1023	-1001	-1000	n	n	33.5	5.067	-5.093
Sl_4	1852	1852	1017	-1003	-999	n	n	34.9	5.076	-5.104
Sl_5	1800	1800	1014	-1004	-1000	n	n	38.5	5.055	-5.108
Sl_6	1786	1785	1011	-1001	-1000	n	n	40.4	5.066	-5.098
Sl_7	1798	1798	1011	-1004	-1000	n	n	37.3	5.083	-5.097
Sl_8	1795	1795	1018	-1002	-1002	n	n	32	5.073	-5.092
Sl_9	1801	1801	1016	-1001	-1002	n	n	35.1	5.09	-5.104
Sl_10	1797	1798	1033	-1001	-1000	n	n	34.7	5.065	-5.104
Sl_11	1795	1796	1019	-1000	-1002	n	n	31.3	5.057	-5.102
Sl_12	1797	0	1013	0	0	n	n	0	-0.022	-0.006
Sl_13	1798	1798	1016	-1002	-1000	n	n	34.3	5.067	-5.102
Sl_14	1793	1793	1016	-1000	-1000	n	n	32.4	5.07	-5.095
Sl_15	1799	1800	1015	-1000	-1001	n	n	28.9	5.068	-5.092
Sl_16	1782	1783	1007	-1002	-1001	n	n	37.7	5.058	-5.099
Sl_17	1798	1798	1011	-1001	-999	n	n	33.3	5.104	-5.094
Sl_18	1796	1796	1017	-1001	-1002	n	n	30.6	5.078	-5.103
Sl_19	1798	1797	1009	-1000	-1001	n	n	34.7	5.07	-5.106
Sl_20	1803	1803	1014	-1002	-1000	n	n	37.6	5.066	-5.11
Sl_21	1799	1799	1010	-1000	-1002	n	n	38.7	5.056	-5.11
Sl_22	1805	1805	1015	-1000	-1001	n	n	33.1	5.066	-5.114
Sl_23	1793	1793	1019	-1000	-1001	n	n	31.2	5.055	-5.096
Sl_24	1789	1788	1018	-1000	-1002	n	n	38.1	5.047	-5.105

Figure 18: Slow control page.

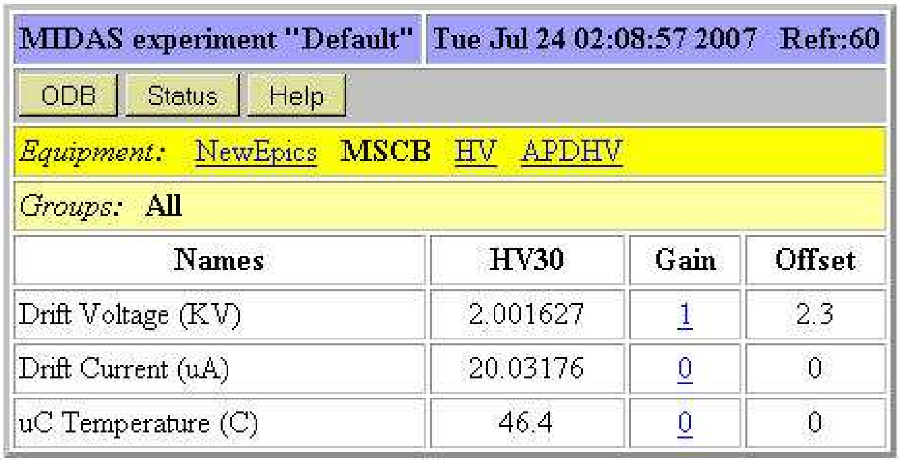
6.12.4 Slow Control page

The page refers to the specific display of a Slow control equipment (see The Equipment structure). In this case the matching names of the equipment variables defined under "Settings/" will be use to compose a table with the "Variables/". Each variable in the table may be editable depending on the follwowing rules

1. If the variable name is defined as "Demand[ ]" or "Output[ ]" or "D_<var_name>" under "Settings/" it will be editable by default.
2. If the variable name is defined under "Settings" and the variable name is also defined under the array "Settings/Editable[ ]" it will be editable.

```
[local:Default:S]/>cd Equipment/MSCB/Settings/
[local:Default:S]Settings>ls
[local:Default:S]Settings>ls
Names
                                 Drift Voltage (KV)
                                 Drift Current (uA)
                                 uC Temperature (C)
DD
Offset
                                 0
                                 1
                                 1
Gain
                                 0
                                 3
                                 4
Editable                         Gain
```

Slow control Equipment page.


MIDAS experiment "Default"
Tue Jul 24 02:08:57 2007 Refr:60
ODB
Status
Help
Equipment: NewEpics **MSCB** HV APDHV
Groups: **All**

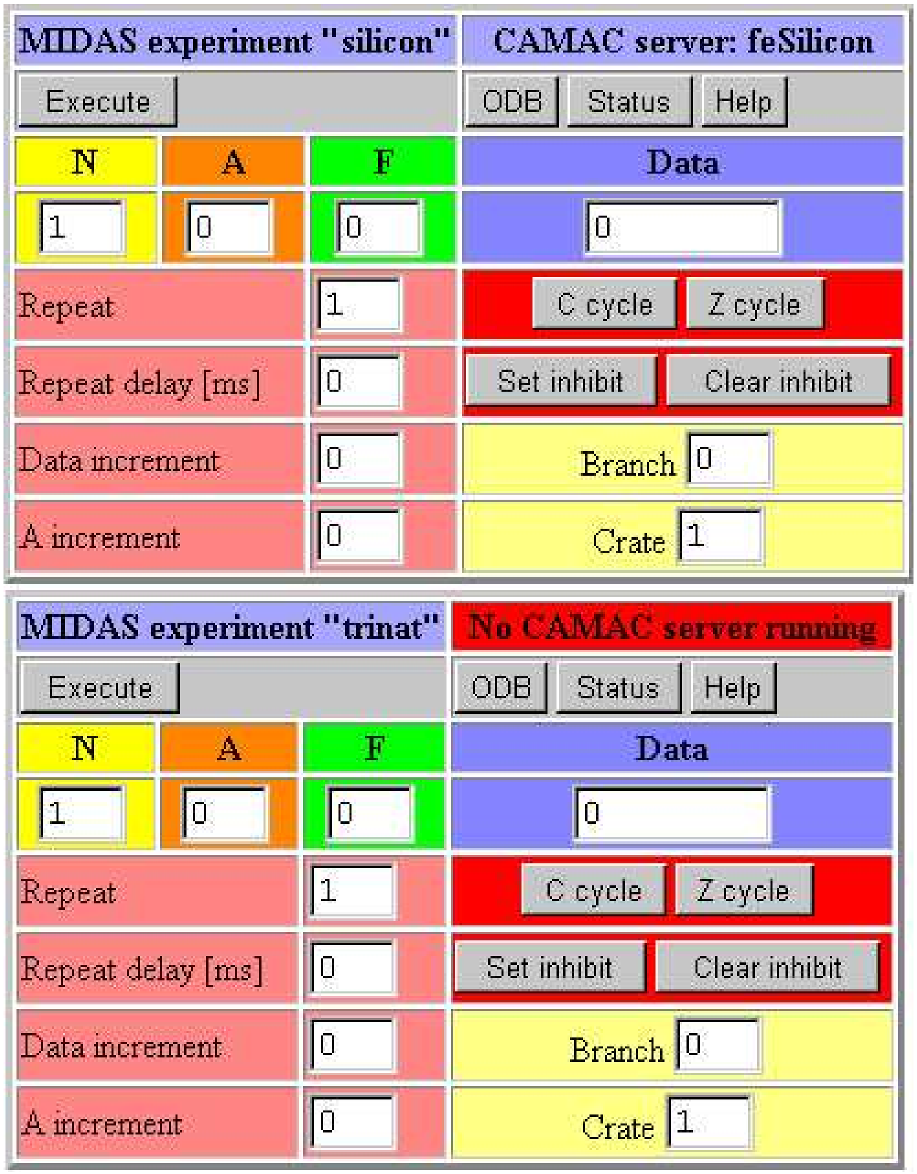
Names	HV30	Gain	Offset
Drift Voltage (KV)	2.001627	1	2.3
Drift Current (uA)	20.03176	0	0
uC Temperature (C)	46.4	0	0

Figure 19: Slow control Equipment page.

6.12.5 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.



MIDAS experiment "silicon"

CAMAC server: feSilicon

N	A	F	Data
1	0	0	0

Repeat	1
Repeat delay [ms]	0
Data increment	0
A increment	0
Branch	0
Crate	1

MIDAS experiment "trinat"

No CAMAC server running

N	A	F	Data
1	0	0	0

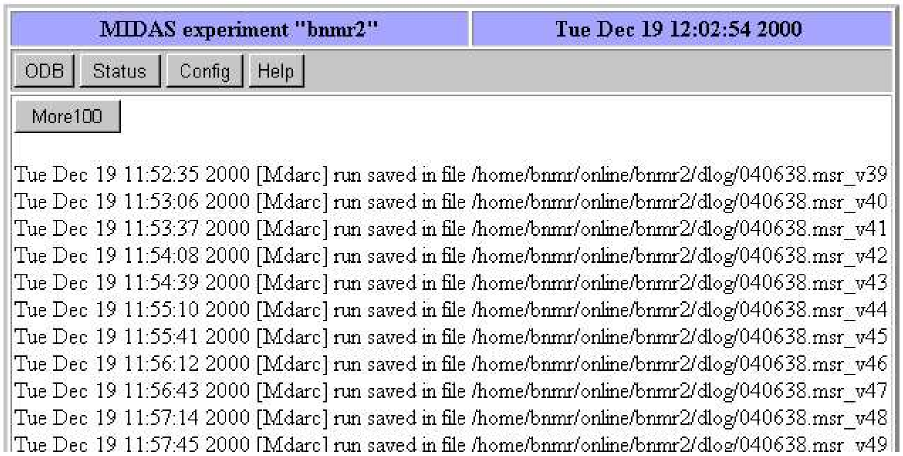
Repeat	1
Repeat delay [ms]	0
Data increment	0
A increment	0
Branch	0
Crate	1

Figure 20: CAMAC command pages.

6.12.6 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.



```
Tue Dec 19 11:52:35 2000 [Mdarc] run saved in file /home/bnmr/online/bnmr2/dlog/040638.msr_v39
Tue Dec 19 11:53:06 2000 [Mdarc] run saved in file /home/bnmr/online/bnmr2/dlog/040638.msr_v40
Tue Dec 19 11:53:37 2000 [Mdarc] run saved in file /home/bnmr/online/bnmr2/dlog/040638.msr_v41
Tue Dec 19 11:54:08 2000 [Mdarc] run saved in file /home/bnmr/online/bnmr2/dlog/040638.msr_v42
Tue Dec 19 11:54:39 2000 [Mdarc] run saved in file /home/bnmr/online/bnmr2/dlog/040638.msr_v43
Tue Dec 19 11:55:10 2000 [Mdarc] run saved in file /home/bnmr/online/bnmr2/dlog/040638.msr_v44
Tue Dec 19 11:55:41 2000 [Mdarc] run saved in file /home/bnmr/online/bnmr2/dlog/040638.msr_v45
Tue Dec 19 11:56:12 2000 [Mdarc] run saved in file /home/bnmr/online/bnmr2/dlog/040638.msr_v46
Tue Dec 19 11:56:43 2000 [Mdarc] run saved in file /home/bnmr/online/bnmr2/dlog/040638.msr_v47
Tue Dec 19 11:57:14 2000 [Mdarc] run saved in file /home/bnmr/online/bnmr2/dlog/040638.msr_v48
Tue Dec 19 11:57:45 2000 [Mdarc] run saved in file /home/bnmr/online/bnmr2/dlog/040638.msr_v49
```

Figure 21: Message page.

6.12.7 Elog page

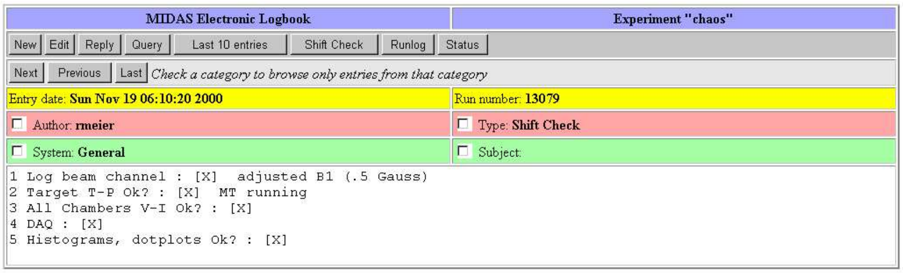
The ELOG page provides access to an electronic logbook. This tool can replace the experimental logbook for daily entries. The main advantage of Elog over paper logbook is the possiblity to access it remotely and provide a general knowledge of the experiment. In the other hand, Elog is not limited strictly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

6.12.7.1 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.



Check a category to browse only entries from that category

Entry date: **Sun Nov 19 06:10:20 2000**

Run number: **13079**

☐ Author: **rmeier**

☐ Type: **Shift Check**

☐ System: **General**

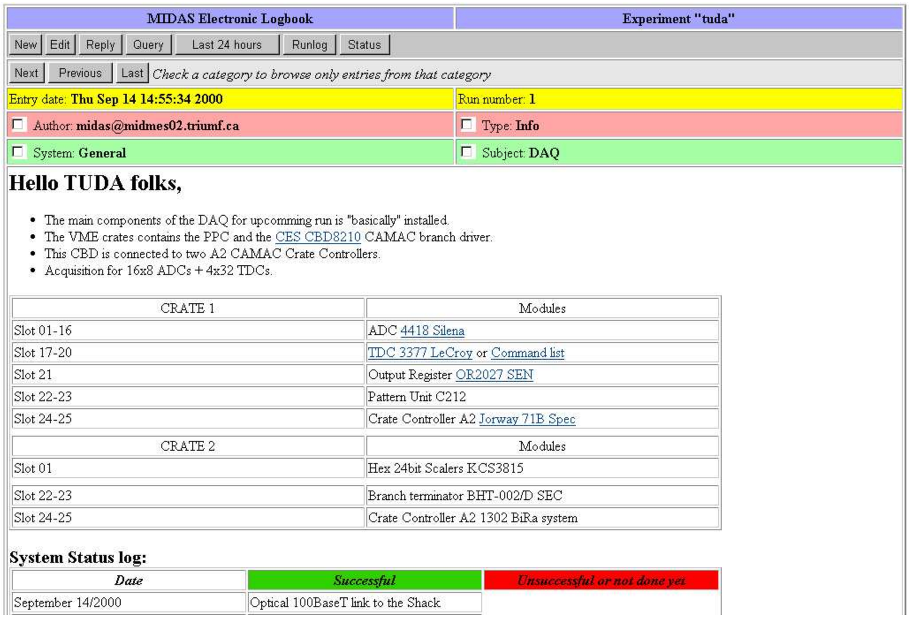
☐ Subject:

```
1 Log beam channel : [X]  adjusted B1 (.5 Gauss)
2 Target T-P Ok? : [X]  MT running
3 All Chambers V-I Ok? : [X]
4 DAQ : [X]
5 Histograms, dotplots Ok? : [X]
```

Figure 22: main Elog page.

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

HTML Elog message.



MIDAS Electronic Logbook	Experiment "tuda"

Check a category to browse only entries from that category

Entry date: **Thu Sep 14 14:55:34 2000**	Run number: **1**
☐ Author: **midas@midmes02.triumf.ca**	☐ Type: **Info**
☐ System: **General**	☐ Subject: **DAQ**

Hello TUDA folks,

- The main components of the DAQ for upcomming run is "basically" installed.
- The VME crates contains the PPC and the CES CBD8210 CAMAC branch driver.
- This CBD is connected to two A2 CAMAC Crate Controllers.
- Acquisition for 16x8 ADCs + 4x32 TDCs.

CRATE 1	Modules
Slot 01-16	ADC 4418 Silena
Slot 17-20	TDC 3377 LeCroy or Command list
Slot 21	Output Register OR2027 SEN
Slot 22-23	Pattern Unit C212
Slot 24-25	Crate Controller A2 Jorway 71B Spec
CRATE 2	Modules
Slot 01	Hex 24bit Scalers KCS3815
Slot 22-23	Branch terminator BHT-002/D SEC
Slot 24-25	Crate Controller A2 1302 BiRa system

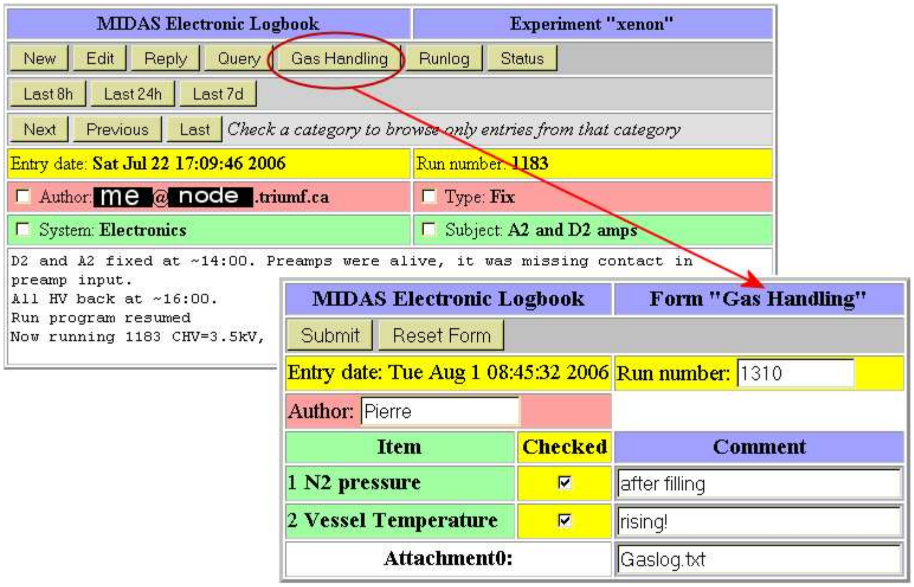
System Status log:

Date	*Successful*	*Unsuccessful or not done yet*
September 14/2000	Optical 100BaseT link to the Shack	

Figure 23: 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.



MIDAS Electronic Logbook | Experiment "xenon"

Check a category to browse only entries from that category

Entry date: **Sat Jul 22 17:09:46 2006** | Run number: **1183**

☐ Author: me @ node .triumf.ca | ☐ Type: **Fix**

☐ System: **Electronics** | ☐ Subject: **A2 and D2 amps**

```
D2 and A2 fixed at ~14:00. Preamps were alive, it was missing contact in
preamp input.
All HV back at ~16:00.
Run program resumed
Now running 1183 CHV=3.5kV,
```

MIDAS Electronic Logbook | **Form "Gas Handling"**

Entry date: Tue Aug 1 08:45:32 2006 | Run number: 1310

Author: Pierre

Item	Checked	Comment
1 N2 pressure	☑	after filling
2 Vessel Temperature	☑	rising!
Attachment0:		Gaslog.txt

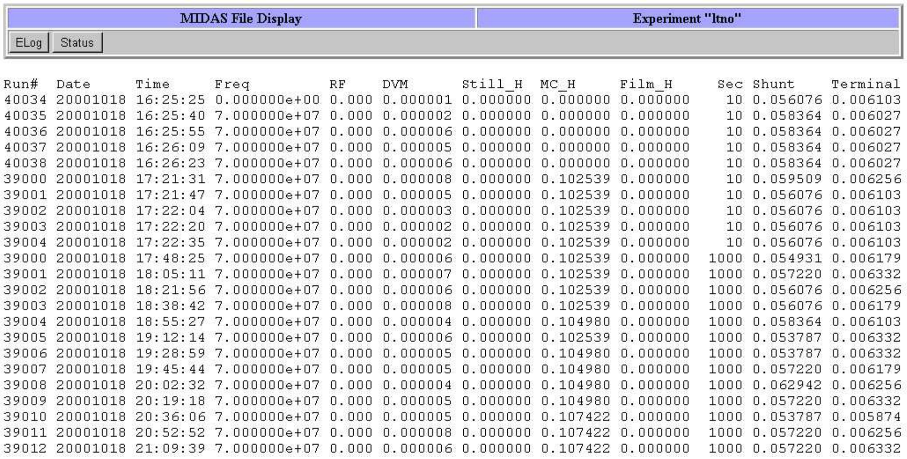
Figure 24: HTML Elog message.

- 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.

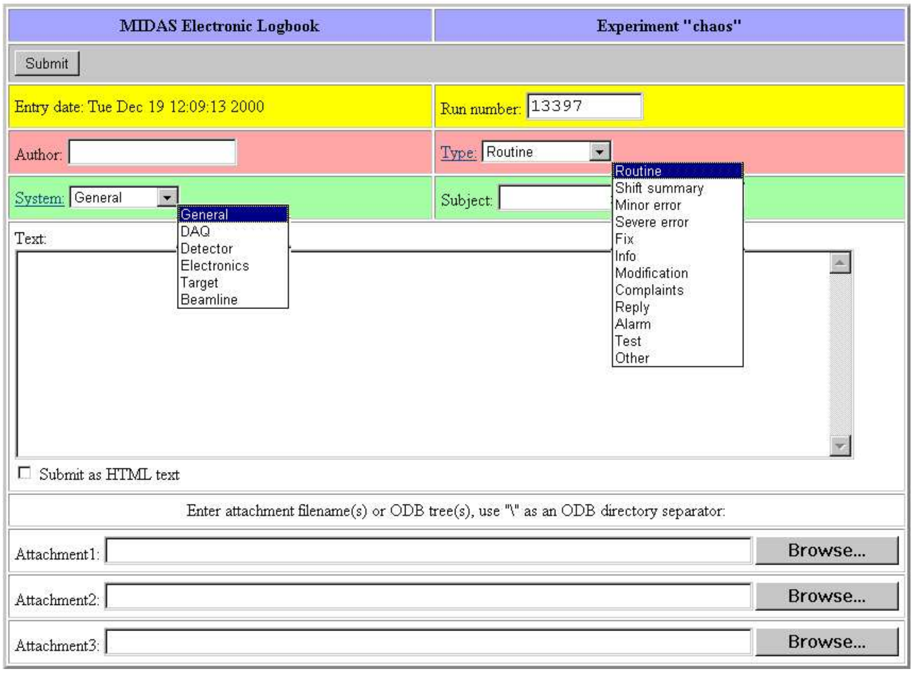


```
Run#  Date     Time     Freq         RF    DVM      Still_H  MC_H     Film_H    Sec Shunt    Terminal
40034 20001018 16:25:25 0.000000e+00 0.000 0.000001 0.000000 0.000000 0.000000   10 0.056076 0.006103
40035 20001018 16:25:40 7.000000e+07 0.000 0.000002 0.000000 0.000000 0.000000   10 0.058364 0.006027
40036 20001018 16:25:55 7.000000e+07 0.000 0.000006 0.000000 0.000000 0.000000   10 0.058364 0.006027
40037 20001018 16:26:09 7.000000e+07 0.000 0.000005 0.000000 0.000000 0.000000   10 0.058364 0.006027
40038 20001018 16:26:23 7.000000e+07 0.000 0.000006 0.000000 0.000000 0.000000   10 0.058364 0.006027
39000 20001018 17:21:31 7.000000e+07 0.000 0.000008 0.000000 0.102539 0.000000   10 0.059509 0.006256
39001 20001018 17:21:47 7.000000e+07 0.000 0.000005 0.000000 0.102539 0.000000   10 0.056076 0.006103
39002 20001018 17:22:04 7.000000e+07 0.000 0.000003 0.000000 0.102539 0.000000   10 0.056076 0.006103
39003 20001018 17:22:20 7.000000e+07 0.000 0.000002 0.000000 0.102539 0.000000   10 0.056076 0.006103
39004 20001018 17:22:35 7.000000e+07 0.000 0.000002 0.000000 0.102539 0.000000   10 0.056076 0.006103
39000 20001018 17:48:25 7.000000e+07 0.000 0.000006 0.000000 0.102539 0.000000 1000 0.054931 0.006179
39001 20001018 18:05:11 7.000000e+07 0.000 0.000007 0.000000 0.102539 0.000000 1000 0.057220 0.006332
39002 20001018 18:21:56 7.000000e+07 0.000 0.000006 0.000000 0.102539 0.000000 1000 0.056076 0.006256
39003 20001018 18:38:42 7.000000e+07 0.000 0.000008 0.000000 0.102539 0.000000 1000 0.056076 0.006179
39004 20001018 18:55:27 7.000000e+07 0.000 0.000004 0.000000 0.104980 0.000000 1000 0.058364 0.006103
39005 20001018 19:12:14 7.000000e+07 0.000 0.000006 0.000000 0.102539 0.000000 1000 0.053787 0.006332
39006 20001018 19:28:59 7.000000e+07 0.000 0.000005 0.000000 0.104980 0.000000 1000 0.053787 0.006332
39007 20001018 19:45:44 7.000000e+07 0.000 0.000005 0.000000 0.104980 0.000000 1000 0.057220 0.006179
39008 20001018 20:02:32 7.000000e+07 0.000 0.000004 0.000000 0.104980 0.000000 1000 0.062942 0.006256
39009 20001018 20:19:18 7.000000e+07 0.000 0.000005 0.000000 0.104980 0.000000 1000 0.057220 0.006332
39010 20001018 20:36:06 7.000000e+07 0.000 0.000005 0.000000 0.107422 0.000000 1000 0.053787 0.005874
39011 20001018 20:52:52 7.000000e+07 0.000 0.000008 0.000000 0.107422 0.000000 1000 0.057220 0.006256
39012 20001018 21:09:39 7.000000e+07 0.000 0.000006 0.000000 0.107422 0.000000 1000 0.057220 0.006332
```

Figure 25: 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.



MIDAS Electronic Logbook	Experiment "chaos"
Entry date: Tue Dec 19 12:09:13 2000	Run number: 13397
Author:	Type: Routine
System: General	Subject:

Text:

☐ Submit as HTML text

Enter attachment filename(s) or ODB tree(s), use "\" as an ODB directory separator:

Attachment1:

Attachment2:

Attachment3:

Figure 26: Elog page, New Elog entry form.

6.12.7.2 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 4032 2007-11-02 17:13:52Z 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 argments 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.

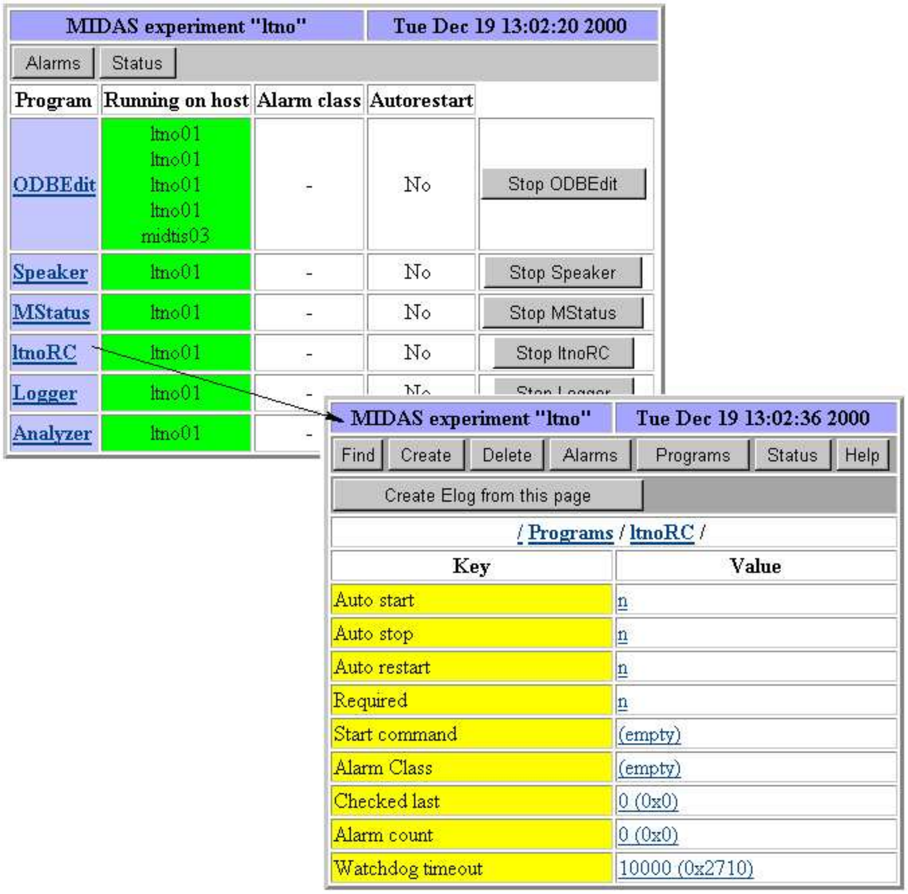
   ```
   NodeB:~> cp internal/elog_logbook/*.log /myexpt/logbook/.
   NodeB:~> cd /myexpt/logbook
   NodeB:~> /installation_elog_dir/elconv
   ```

6.12.8 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.



MIDAS experiment "ltno" Tue Dec 19 13:02:20 2000

Program	Running on host	Alarm class	Autorestart
ODBEdit	ltno01 ltno01 ltno01 ltno01 midtis03	-	No
Speaker	ltno01	-	No
MStatus	ltno01	-	No
ltnoRC	ltno01	-	No
Logger	ltno01	-	No
Analyzer	ltno01	-	

MIDAS experiment "ltno" Tue Dec 19 13:02:36 2000

/ Programs / ltnoRC /

Key	Value
Auto start	n
Auto stop	n
Auto restart	n
Required	n
Start command	(empty)
Alarm Class	(empty)
Checked last	0 (0x0)
Alarm count	0 (0x0)
Watchdog timeout	10000 (0x2710)

Figure 27: Program page.

6.12.9 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.

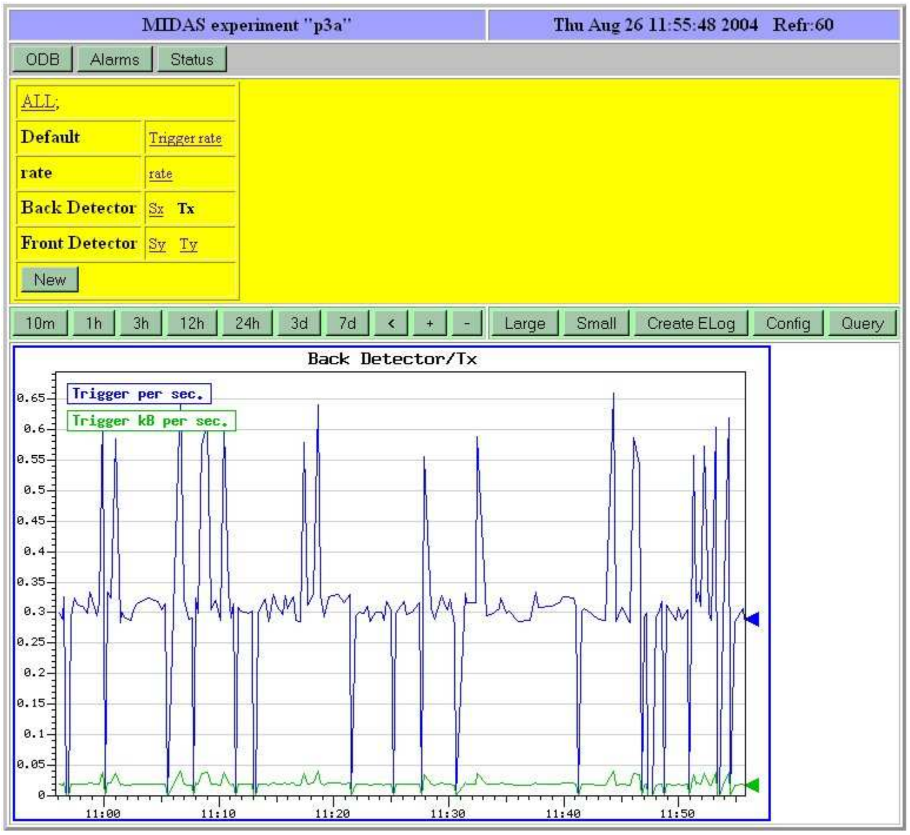
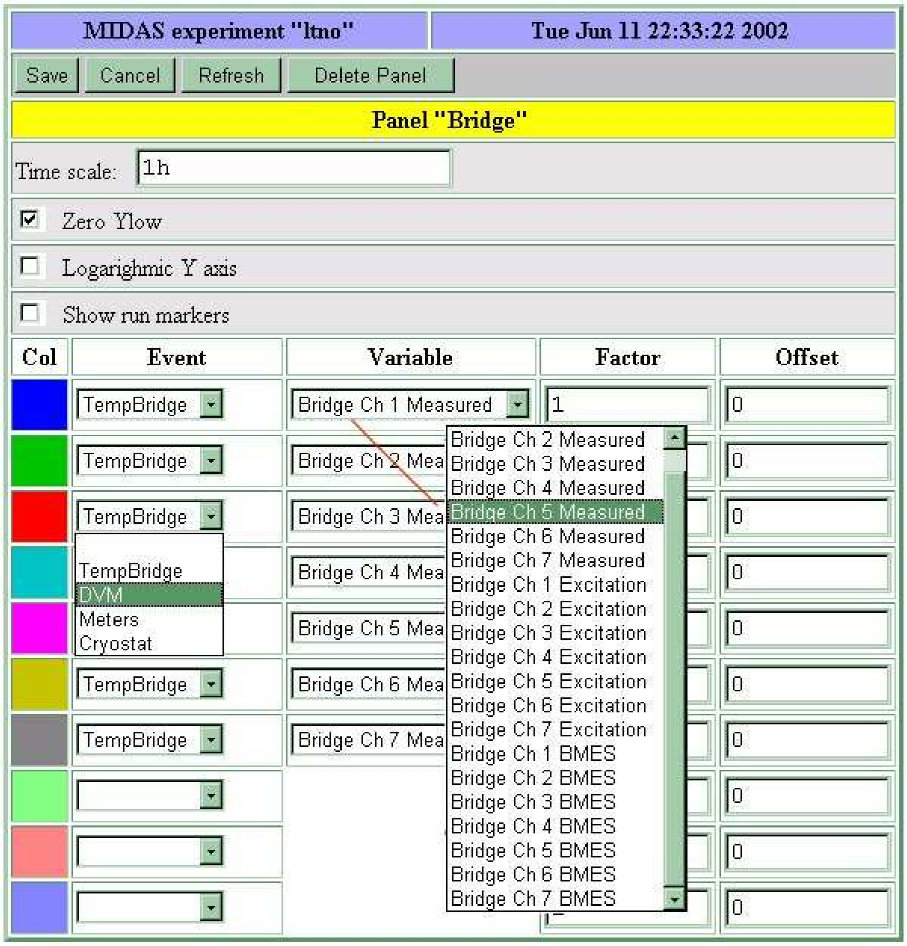


Figure 28: History page.

History channel selection Page.



MIDAS experiment "ltno" — Tue Jun 11 22:33:22 2002

Panel "Bridge"

Time scale: 1h

☑ Zero Ylow

☐ Logarighmic Y axis

☐ Show run markers

Col	Event	Variable	Factor	Offset
🟦	TempBridge	Bridge Ch 1 Measured	1	0
🟩	TempBridge	Bridge Ch 2 Mea		0
🟥	TempBridge	Bridge Ch 3 Mea		0
🟦		Bridge Ch 4 Mea		0
🟪		Bridge Ch 5 Mea		0
🟨	TempBridge	Bridge Ch 6 Mea		0
⬛	TempBridge	Bridge Ch 7 Mea		0
🟩				0
🟥				0
🟦				0

Figure 29: History channel selection Page.

6.12.10 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.

6.12.11 Custom page

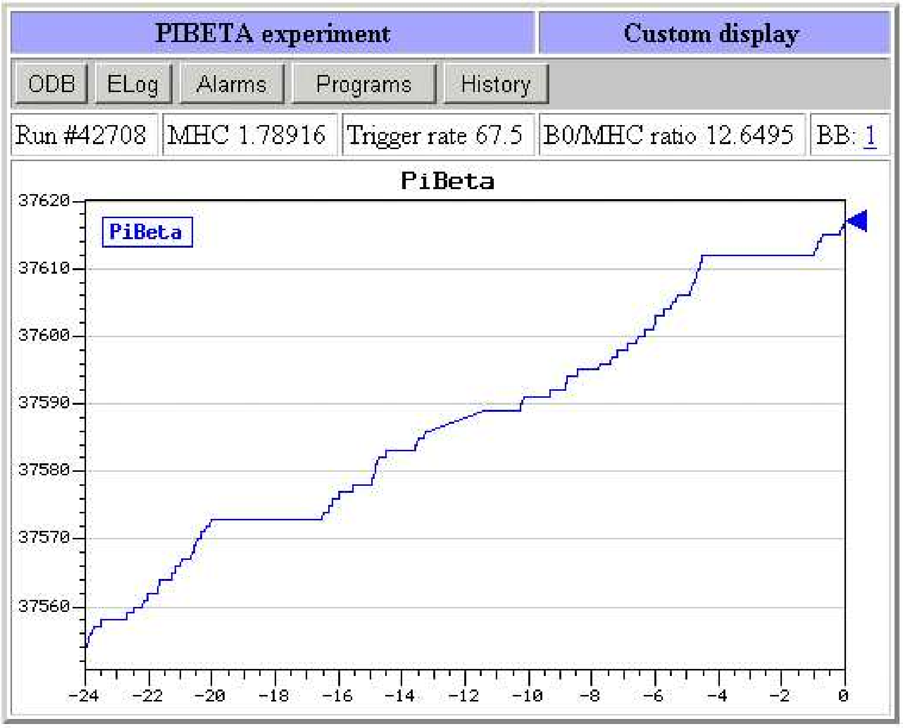
The Custom page is available since version 1.8.3. It has been improved during version 1.9.5 (mhttpd.c CVS-1.288).

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.

6.12.11.1 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.



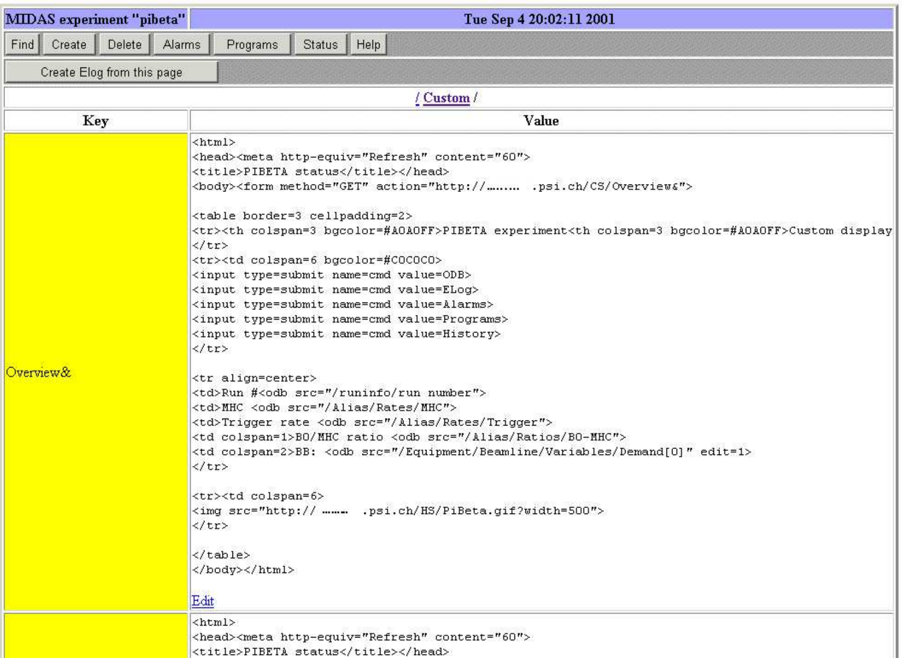
Run #42708 | MHC 1.78916 | Trigger rate 67.5 | B0/MHC ratio 12.6495 | BB: 1

Figure 30: 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.



Overview&

```
<html>
<head><meta http-equiv="Refresh" content="60">
<title>PIBETA status</title></head>
<body><form method="GET" action="http://........ .psi.ch/CS/Overview&">

<table border=3 cellpadding=2>
<tr><th colspan=3 bgcolor=#A0A0FF>PIBETA experiment<th colspan=3 bgcolor=#A0A0FF>Custom display
</tr>
<tr><td colspan=6 bgcolor=#C0C0C0>
<input type=submit name=cmd value=ODB>
<input type=submit name=cmd value=ELog>
<input type=submit name=cmd value=Alarms>
<input type=submit name=cmd value=Programs>
<input type=submit name=cmd value=History>
</tr>

<tr align=center>
<td>Run #<odb src="/runinfo/run number">
<td>MHC <odb src="/Alias/Rates/MHC">
<td>Trigger rate <odb src="/Alias/Rates/Trigger">
<td colspan=1>B0/MHC ratio <odb src="/Alias/Ratios/B0-MHC">
<td colspan=2>BB: <odb src="/Equipment/Beamline/Variables/Demand[0]" edit=1>
</tr>

<tr><td colspan=6>
<img src="http:// ....... .psi.ch/HS/PiBeta.gif?width=500">
</tr>

</table>
</body></html>
```

```
<html>
<head><meta http-equiv="Refresh" content="60">
<title>PIBETA status</title></head>
```

Figure 31: 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>
```

6.12.11.2 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:

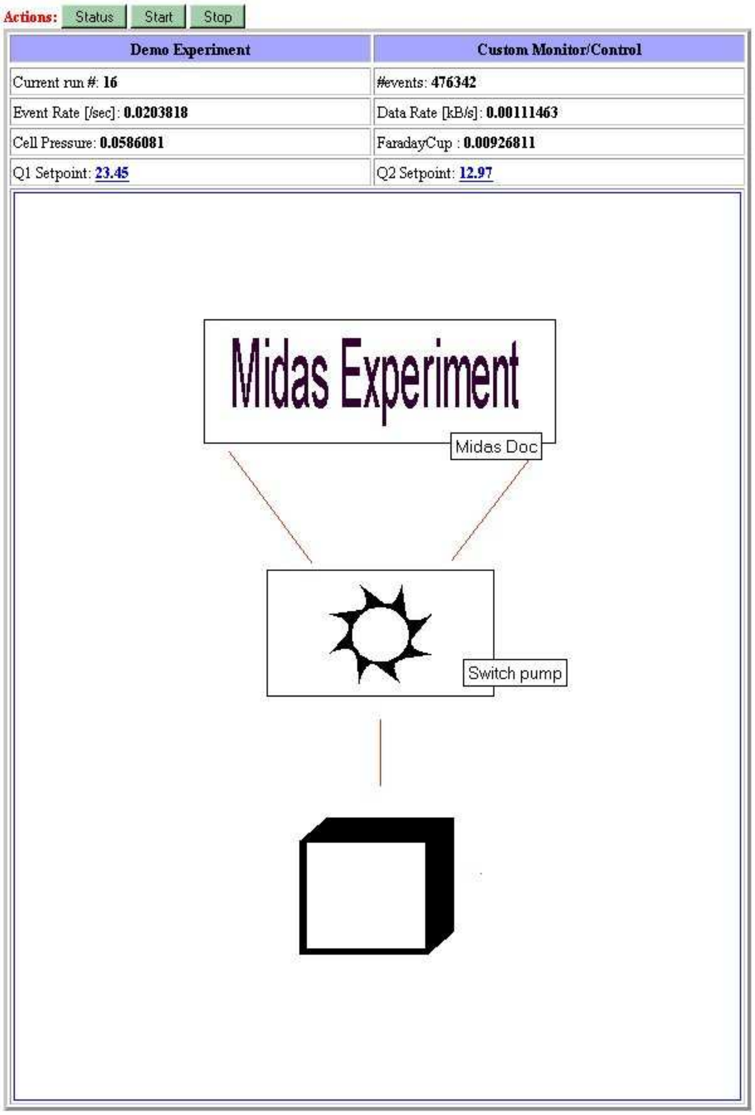
```
[local:Default:Stopped]/>cd /Custom
[local:Default:Stopped]/Custom>create string Myexpt&
String length [32]: 256
[local:Default:Stopped]/Custom>set Myexpt& \midas\examples\custom\myexpt.html
```

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.



Demo Experiment	Custom Monitor/Control
Current run #: **16**	#events: **476342**
Event Rate [/sec]: **0.0203818**	Data Rate [kB/s]: **0.00111463**
Cell Pressure: **0.0586081**	FaradayCup : **0.00926811**
Q1 Setpoint: **23.45**	Q2 Setpoint: **12.97**

Figure 32: 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
```

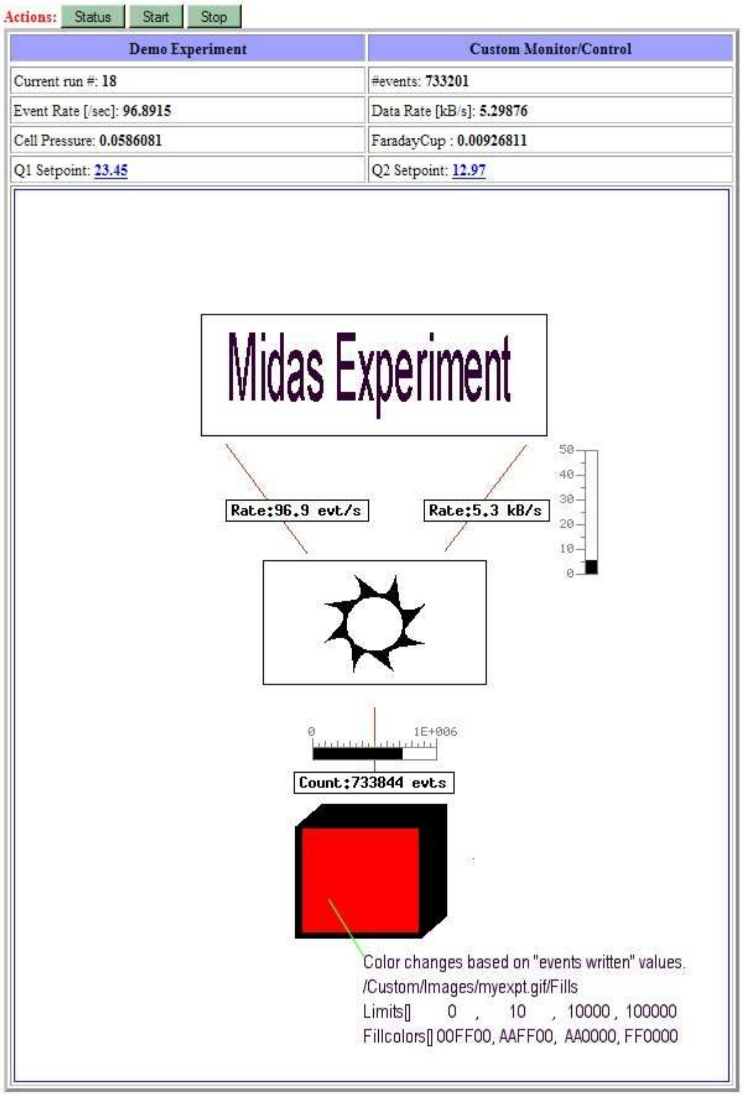
```
>>>>>>>> Refresh web page <<<<<<<<

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.



Demo Experiment	Custom Monitor/Control
Current run #: **18**	#events: **733201**
Event Rate [/sec]: **96.8915**	Data Rate [kB/s]: **5.29876**
Cell Pressure: **0.0586081**	FaradayCup : **0.00926811**
Q1 Setpoint: 23.45	Q2 Setpoint: 12.97

Color changes based on "events written" values.
/Custom/Images/myexpt.gif/Fills
Limits[] 0 , 10 , 10000 , 100000
Fillcolors[] 00FF00, AAFF00, AA0000, FF0000

Figure 33: Custom web external to html document.

6.12.11.3 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:

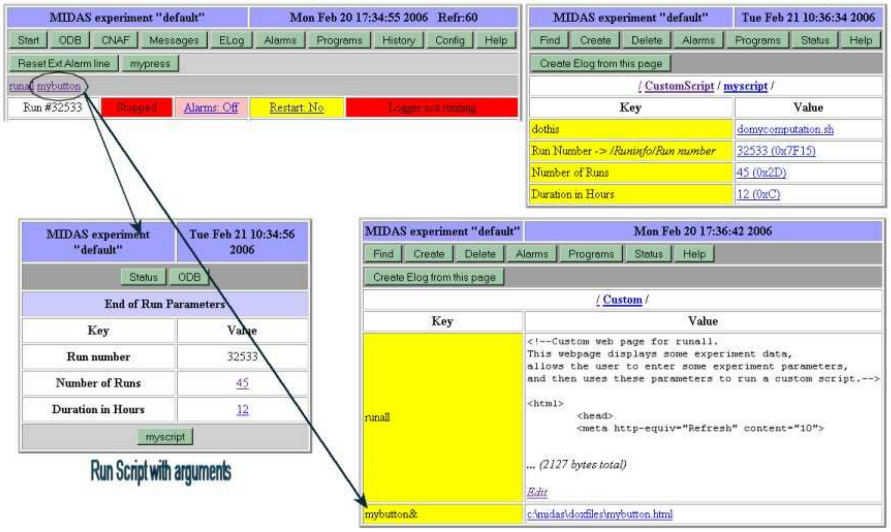
- Create the mybutton.html code as described in the Internal HTML document..
- Create a new custom (internal or External) in ODB under **/Custom/mybutton&**
- Create a new custom script (with argument as described in the myscript.html) in ODB under **/CustomScript/myscript&**

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.



Key	Value
dothis	domycomputation.sh
Run Number -> */Runinfo/Run number*	32533 (0x7F15)
Number of Runs	45 (0x2D)
Duration in Hours	12 (0xC)

Key	Value
Run number	32533
Number of Runs	45
Duration in Hours	12

Run Script with arguments

Key	Value
runall	see below
mybutton&	c:\midas\doxfiles\mybutton.html

runall:

```
<!--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">
```

... *(2127 bytes total)*

Edit

Figure 34: 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.

- Software version: 2.0.0

- **[>2.0.0] Nov 02/2007**
  - mlogger task supports zip files for MIDAS format.
  - New environment variable MIDAS_MAX_EVENT_SIZE.
  - lazylogger task new features (remark).
  - Midas Code and Libraries **alarm, history, elog** functions have been placed in dedicate files.
  - Slow Control page new editable field for web display.

- **[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).
  - Midas Ring Buffer Functions (rb_xxx) for multi-threading and cascading data transfer.
- **[<2.0.0]**
  - Unsupported 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 predefined 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

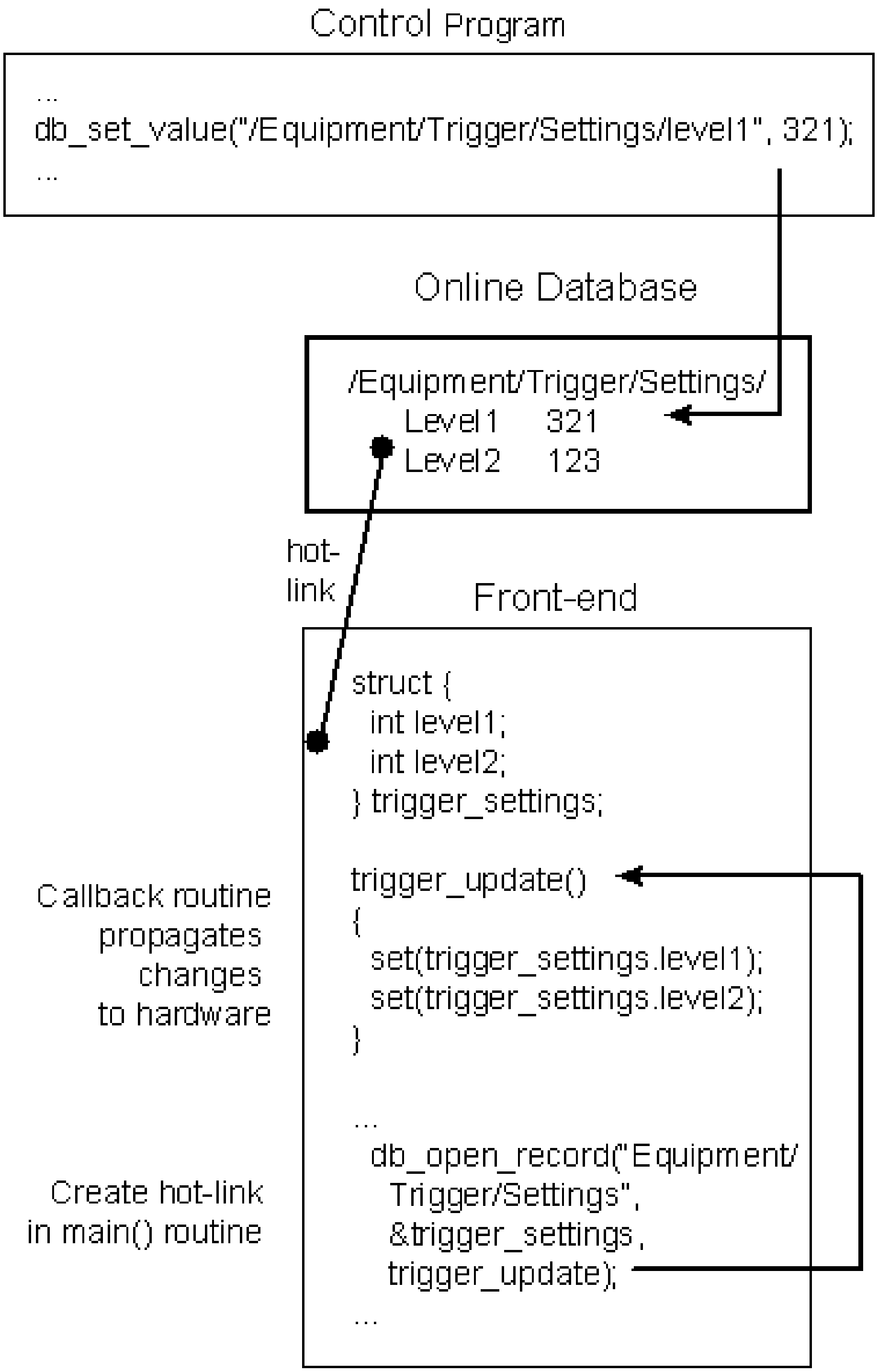


Figure 35: HotLink process

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]

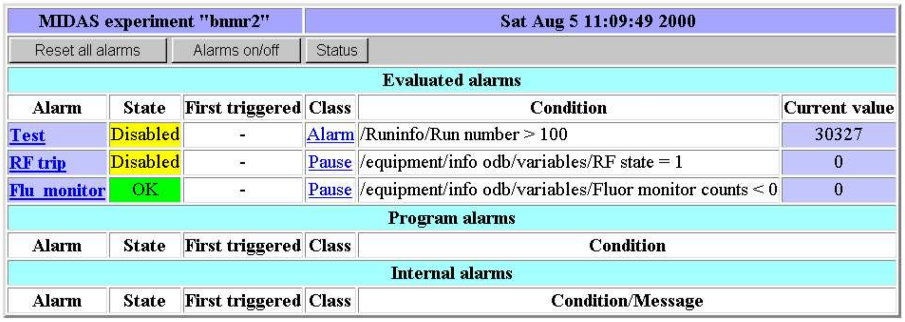
6.14.17 Alarm System

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

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
Flu_monitor	OK	-	Pause	/equipment/info odb/variables/Fluor monitor counts < 0	0

Program alarms

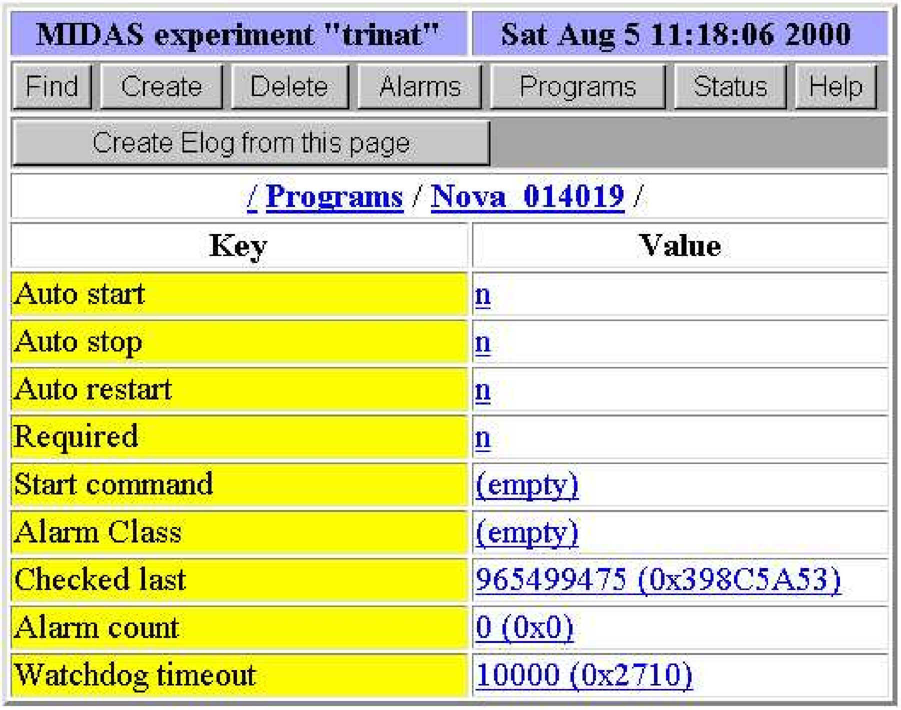
Alarm	State	First triggered	Class	Condition

Internal alarms

Alarm	State	First triggered	Class	Condition/Message

Figure 36: Midas Web Alarm setting display

Midas Web Alarm setting display



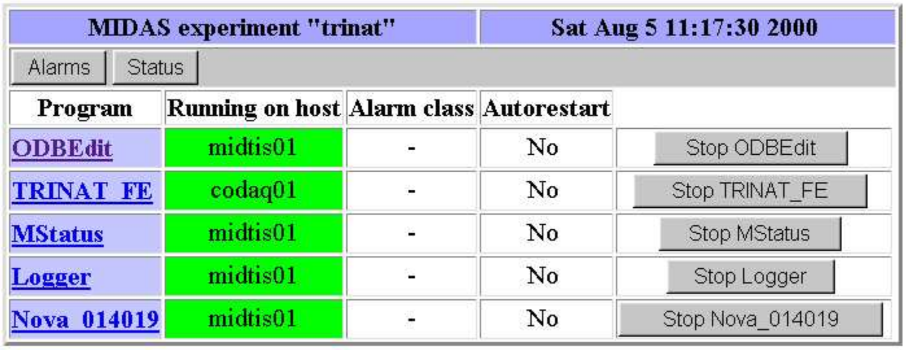
MIDAS experiment "trinat" Sat Aug 5 11:18:06 2000

/ 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 37: Midas Web Alarm setting display

Midas Web Alarm Program status display



MIDAS experiment "trinat"				Sat Aug 5 11:17:30 2000
Alarms Status				
Program	Running on host	Alarm class	Autorestart	
ODBEdit	midtis01	-	No	Stop ODBEdit
TRINAT_FE	codaq01	-	No	Stop TRINAT_FE
MStatus	midtis01	-	No	Stop MStatus
Logger	midtis01	-	No	Stop Logger
Nova_014019	midtis01	-	No	Stop Nova_014019

Figure 38: 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 some 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
  ```

1. **Compilation & Build:** Previously we did supplied RPM or compress package which included an initial binary version of all the midas application for linux. Using now SVN and its ability to generate tarball directly from the svn/web interface, compilation and local build is required.

   Once the midas package has been extracted into your desired directory, define the environment variable **MIDASSYS** to point to this directory. run **make** to produce all the midas core applications. This will create a directory under MIDASSYS based on the host type under which a lib and bin will contain respectively the objects/library and the application binaries.

   The compilation and link will try to generate the **rmidas** application which requires **ROOT** to be installed. The application **mana** will also be compiled for **HBOOK** and **ROOT**. Look in the make listing below for the HAVE_HBOOK, HAVE_ROOT.

   ```
    > setenv MIDASSYS /home/<mydir>/midas
    or
    > export MIDASSYS=/home/<mydir>/midas

    > cd $MIDASSYS
    > 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
```

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. **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.
  - From version >2.0.0
    - mlogger implements compression such that the data file name can be given as <**file.mid.gz**>

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.

- **Remarks >2.0.0**
  - While lazylogger was developed specifically for tape device, it also supports data file transfer to FTP repository system. Improvement towards a more generic support has been done which includes:
    - dynamic directory destination based on run number or date.
    - compression copy.
    - "stay behind=0" support.
    - Script copy support.

- **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
    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 864.020
    Fri Mar 24 15:07:35 2000 [Lazy_Tape] 8352 (rm:25854ms) /scr0/spring2000/run08352.ybs file
    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
    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
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.

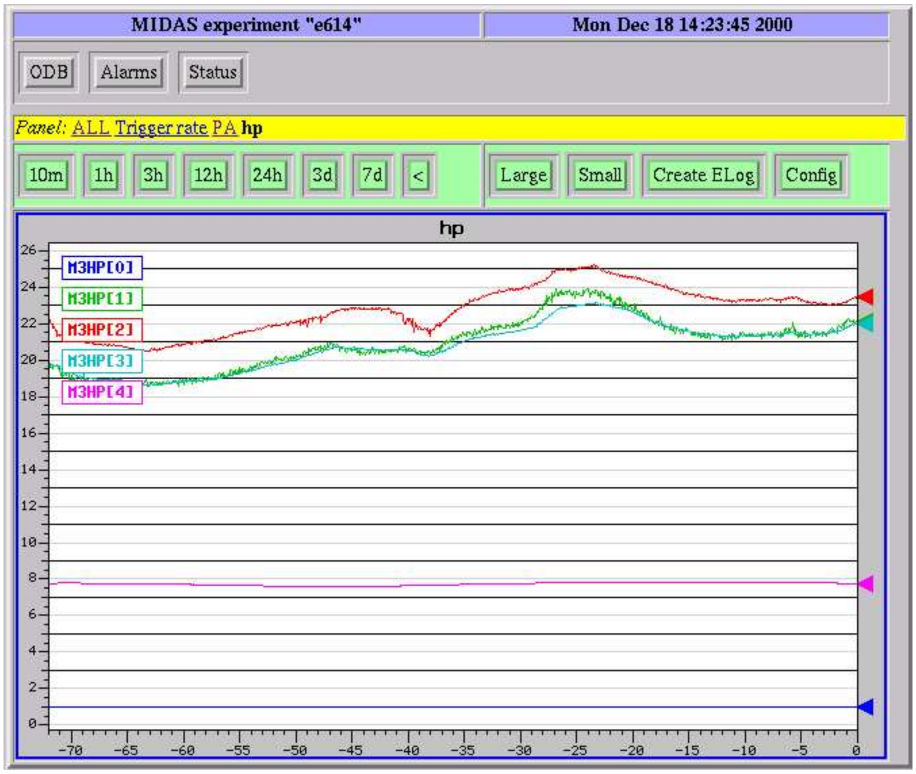


Figure 39: Midas Web History display.

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).

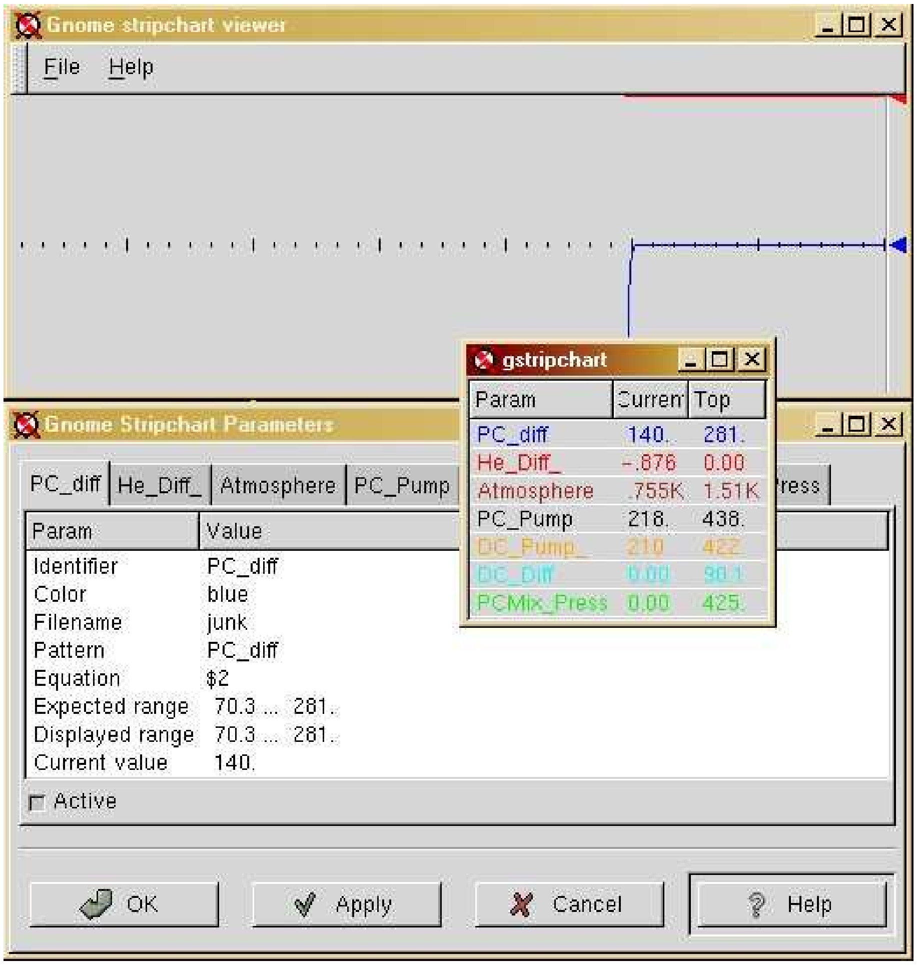
- **Arguments**
  - [-mhist] : start stripchart for Midas history data.
- **Usage** : stripchart <-options> <config-file> -mhist: (look at history file -default) -dmhist: debug mhist -debug: debug stripchart -config_file: see mchart_task

```
> stripchart.tcl -debug
> stripchart.tcl
```

- **Example**

```
> stripchart.tcl -h
```

gstripchart display with parameters and data pop-up.



gstripchart

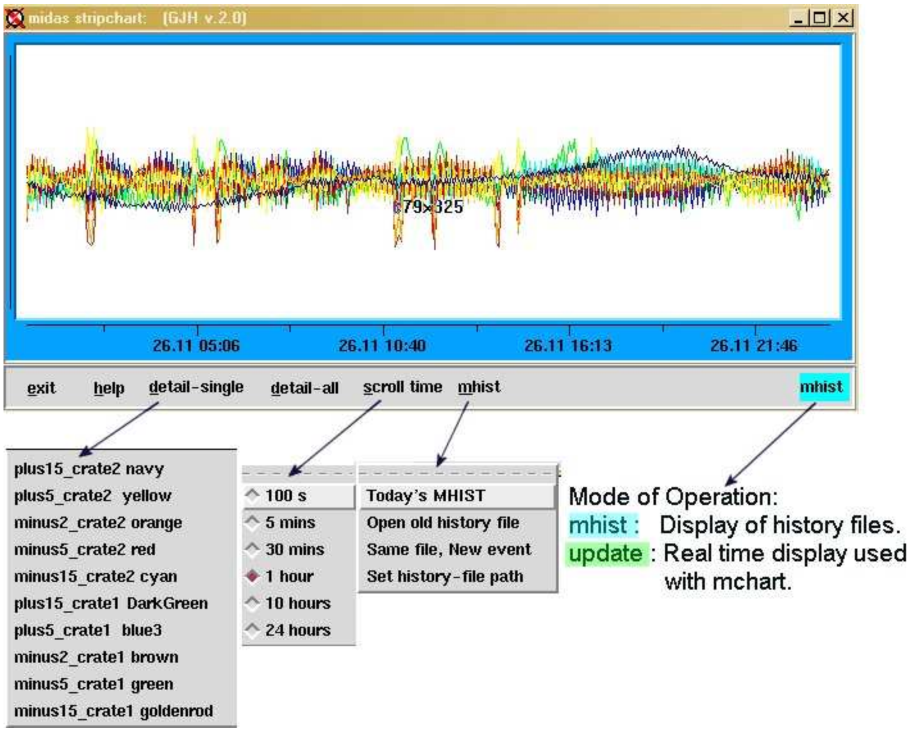
Param	Curren	Top
PC_diff	140.	281.
He_Diff_	-.876	0.00
Atmosphere	.755K	1.51K
PC_Pump	218.	438.
DC_Pump_	210	422
DC_Diff	0.00	90.1
PCMix_Press	0.00	425.

Param	Value
Identifier	PC_diff
Color	blue
Filename	junk
Pattern	PC_diff
Equation	$2
Expected range	70.3 ... 281.
Displayed range	70.3 ... 281.
Current value	140.

☐ Active

Figure 40: gstripchart display with parameters and data pop-up.

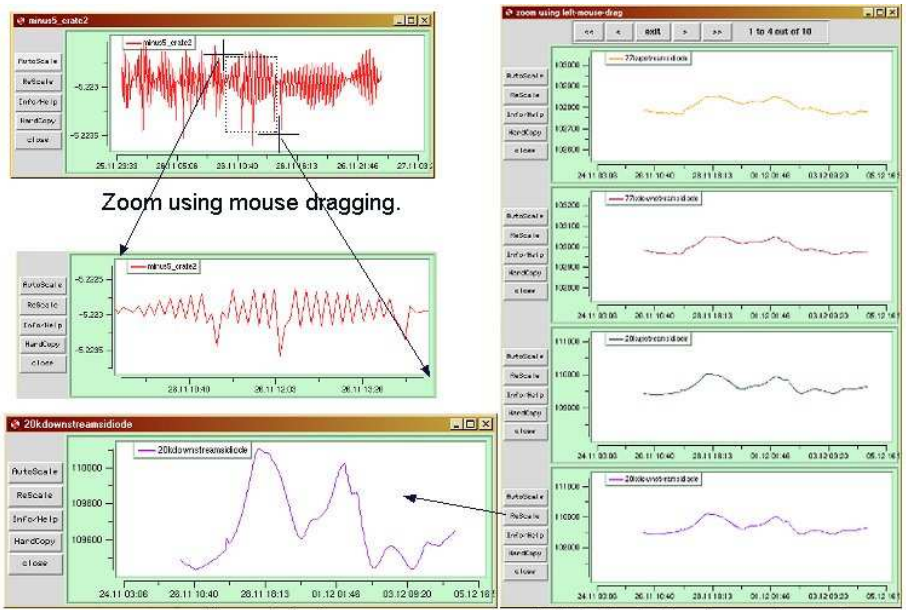
stripchart.tcl mhist mode: main window with pull-downs.



Mode of Operation:
mhist : Display of history files.
update : Real time display used with mchart.

Figure 41: stripchart.tcl mhist mode: main window with pull-downs.

stripchart.tcl Online data, running in conjunction with mchart



Zoom using mouse dragging.

Rescale button

Detail-all selection from main window:
- Display 4 graphs per page.
- Page selection on top of window.

Figure 42: 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.

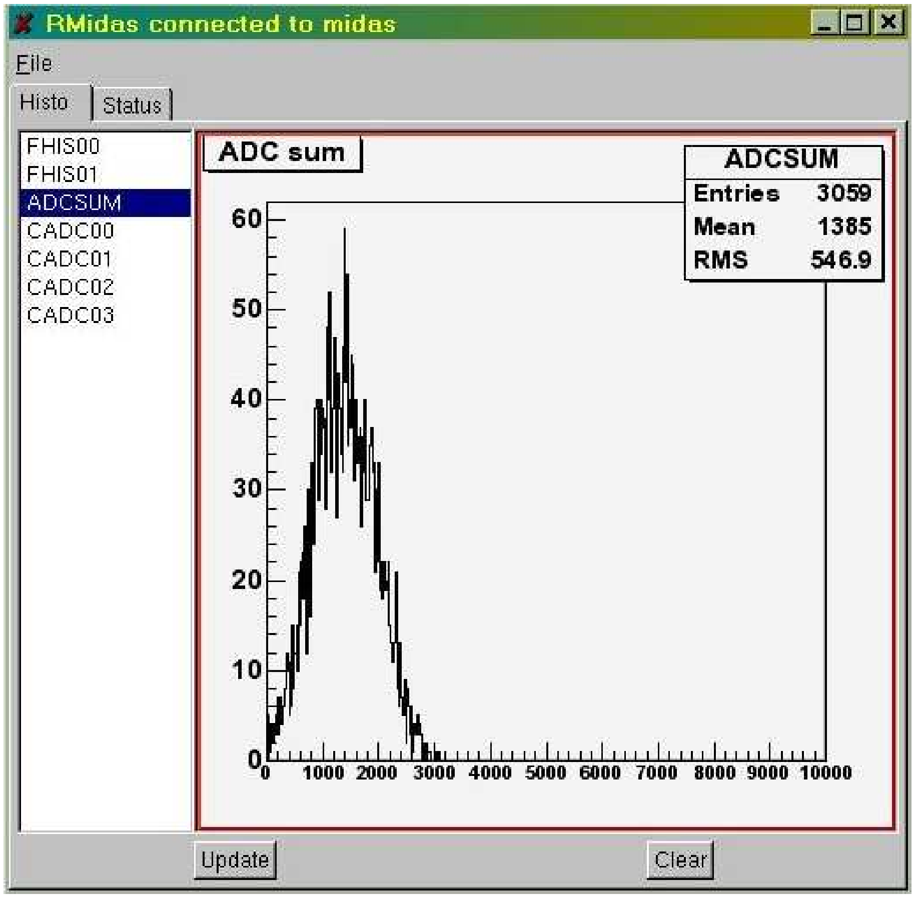


Figure 43: rmidas display sample. Using the example/experiment/ demo setup.

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**