Midas Reference Manual

1.9.5-0

Generated by Doxygen 1.3.9.1

Mon Nov 15 22:52:09 2004

Contents

1 Midas documentation

Welcome to the world of Midas.

1.1 Introduction

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

If you think Midas can help you for your projects and you want to get more info on it, feel free to browse through the following Web sites: `Switzerland` , `Canada`

Chapters are:

- 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.
- 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 CAMAC standard

Modules

- group Camac Functions (camxxx)

2.2 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 INLINE void EXPRT 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)

2.2.1 Function Documentation

2.2.1.1 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

2.2.1.2 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().

2.2.1.3 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

2.2.1.4 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

2.2.1.5 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

2.2.1.6 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

2.2.1.7 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

2.2.1.8 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().

2.2.1.9 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

2.2.1.10 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().

2.2.1.11 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().

2.2.1.12 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

2.2.1.13 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

2.2.1.14 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

2.2.1.15 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

2.2.1.16 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

2.2.1.17 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().

2.2.1.18 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

2.2.1.19 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

2.2.1.20 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

2.2.1.21 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

2.2.1.22 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

2.2.1.23 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

2.2.1.24 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

2.2.1.25 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

2.2.1.26 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().

2.2.1.27 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().

2.2.1.28 EXTERNAL INLINE void EXPRT cam_exit (void)

Close CAMAC accesss.

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

Clear Crate inhibit.

Parameters:
c crate number (0..)

Returns:
void

Referenced by ccci().

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

Set Crate inhibit.

Parameters:
c crate number (0..)

Returns:
void

Referenced by ccci().

2.2.1.31 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().

2.2.1.32 EXTERNAL INLINE int EXPRT cam_init (void)

Initialize CAMAC access.

Returns:
1: success

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

2.2.1.33 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

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

Attach service routine to LAM of speci£c crate and station.

Parameters:

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

Returns:

void

Referenced by cclnk().

2.2.1.35 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().

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

Disables interrupts in speci£c crate

Parameters:
c crate number (0..)

Returns:
void

Referenced by cccd().

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

Enable interrupts in speci£c crate

Parameters:
c crate number (0..)

Returns:
void

Referenced by cccd(), and ccrgl().

2.2.1.38 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().

2.2.1.39 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().

2.2.1.40 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

2.2.1.41 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().

2.2.1.42 EXTERNAL INLINE void EXPRT 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().

2.2.1.43 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

2.2.1.44 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().

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

Crate presence check.

Parameters:

c crate number (0..)

Returns:

0:Success, -1:No CAMAC response

2.2.1.46 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().

2.2.1.47 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

2.2.1.48 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

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

Same as cam16i()

2.2.1.50 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().

2.3 The midas.h & midas.c

Modules

- group Midas De£ne
- group Midas Macros
- group Midas Error de£nition
- 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 History Functions (hs_xxx)
- group Midas Elog Functions (el_xxx)
- group Midas Alarm Functions (al_xxx)
- group Midas Dual Buffer Memory Functions (dm_xxx)

De£nes

- #de£ne TAPE_BUFFER_SIZE 0x8000
- #de£ne NET_TCP_SIZE 0xFFFF
- #de£ne OPT_TCP_SIZE 8192
- #de£ne NET_UDP_SIZE 8192
- #de£ne EVENT_BUFFER_SIZE 0x100000
- #de£ne EVENT_BUFFER_NAME "SYSTEM"
- #de£ne MAX_EVENT_SIZE 0x80000
- #de£ne DEFAULT_EVENT_BUFFER_SIZE 0x200000;
- #de£ne DEFAULT_ODB_SIZE 0x100000
- #de£ne NAME_LENGTH 32
- #de£ne HOST_NAME_LENGTH 256
- #de£ne MAX_CLIENTS 64
- #de£ne MAX_EVENT_REQUESTS 10
- #de£ne MAX_OPEN_RECORDS 256
- #de£ne MAX_ODB_PATH 256
- #de£ne MAX_EXPERIMENT 32
- #de£ne BANKLIST_MAX 64
- #de£ne STRING_BANKLIST_MAX BANKLIST_MAX ∗ 4
- #de£ne DEFAULT_RPC_TIMEOUT 10000
- #de£ne DEFAULT_WATCHDOG_TIMEOUT 10000
- #de£ne CH_BS 8
- #de£ne LAM_SOURCE(c, s) (c<<24 | ((s) & 0xFFFFFF))
- #de£ne LAM_STATION(s) (1<<(s-1))
- #de£ne LAM_SOURCE_CRATE(c) (c>>24)
- #de£ne LAM_SOURCE_STATION(s) ((s) & 0xFFFFFF)
- #de£ne CNAF 0x1
- #de£ne ANA_CONTINUE 1

Variables

- HNDLE _hKeyClient = 0

2.3.1 De£ne Documentation

2.3.1.1 #de£ne ANA_CONTINUE 1

dox***

De£nition at line 1160 of £le midas.h.

2.3.1.2 #de£ne ANA_SKIP 0

De£nition at line 1161 of £le midas.h.

2.3.1.3 #de£ne BANKLIST_MAX 64

max # of banks in event

De£nition at line 674 of £le midas.h.

Referenced by bk_list().

2.3.1.4 #de£ne CH_BS 8

special characters

De£nition at line 814 of £le midas.h.

2.3.1.5 #de£ne CH_CR 13

De£nition at line 816 of £le midas.h.

2.3.1.6 #de£ne CH_DELETE (CH_EXT+2)

De£nition at line 822 of £le midas.h.

2.3.1.7 #de£ne CH_DOWN (CH_EXT+7)

De£nition at line 827 of £le midas.h.

2.3.1.8 #de£ne CH_END (CH_EXT+3)

De£nition at line 823 of £le midas.h.

2.3.1.9 #de£ne CH_EXT 0x100

De£nition at line 818 of £le midas.h.

2.3.1.10 #de£ne CH_HOME (CH_EXT+0)

De£nition at line 820 of £le midas.h.

2.3.1.11 #de£ne CH_INSERT (CH_EXT+1)

De£nition at line 821 of £le midas.h.

2.3.1.12 #de£ne CH_LEFT (CH_EXT+9)

De£nition at line 829 of £le midas.h.

2.3.1.13 #de£ne CH_PDOWN (CH_EXT+5)

De£nition at line 825 of £le midas.h.

2.3.1.14 #de£ne CH_PUP (CH_EXT+4)

De£nition at line 824 of £le midas.h.

2.3.1.15 #de£ne CH_RIGHT (CH_EXT+8)

De£nition at line 828 of £le midas.h.

2.3.1.16 #de£ne CH_TAB 9

De£nition at line 815 of £le midas.h.

2.3.1.17 #de£ne CH_UP (CH_EXT+6)

De£nition at line 826 of £le midas.h.

2.3.1.18 #de£ne CNAF 0x1

CNAF commands

De£nition at line 859 of £le midas.h.

2.3.1.19 #de£ne CNAF_CRATE_CLEAR 0x102

De£nition at line 864 of £le midas.h.

2.3.1.20 #de£ne CNAF_CRATE_ZINIT 0x103

De£nition at line 865 of £le midas.h.

2.3.1.21 #de£ne CNAF_INHIBIT_CLEAR 0x101

De£nition at line 863 of £le midas.h.

2.3.1.22 #de£ne CNAF_INHIBIT_SET 0x100

De£nition at line 862 of £le midas.h.

2.3.1.23 #de£ne CNAF_nQ 0x2

De£nition at line 860 of £le midas.h.

2.3.1.24 #de£ne CNAF_TEST 0x110

De£nition at line 866 of £le midas.h.

2.3.1.25 #de£ne DATABASE_VERSION 2

De£nition at line 493 of £le midas.h.

Referenced by db_open_database().

2.3.1.26 #de£ne DEFAULT_EVENT_BUFFER_SIZE 0x200000;

2M

De£nition at line 664 of £le midas.h.

2.3.1.27 #de£ne DEFAULT_ODB_SIZE 0x100000

online database 1M

De£nition at line 665 of £le midas.h.

Referenced by cm_connect_experiment(), and main().

2.3.1.28 #de£ne DEFAULT_RPC_TIMEOUT 10000

Timeouts [ms]

De£nition at line 682 of £le midas.h.

2.3.1.29 #de£ne DEFAULT_WATCHDOG_TIMEOUT 10000

Watchdog

De£nition at line 685 of £le midas.h.

Referenced by cm_connect_experiment().

2.3.1.30 #de£ne EVENT_BUFFER_NAME "SYSTEM"

buffer name for commands

De£nition at line 662 of £le midas.h.

2.3.1.31 #de£ne EVENT_BUFFER_SIZE 0x100000

buffer used for events

De£nition at line 661 of £le midas.h.

Referenced by register_equipment(), and source_booking().

2.3.1.32 #de£ne HOST_NAME_LENGTH 256

length of TCP/IP names

De£nition at line 668 of £le midas.h.

Referenced by cm_set_client_info().

2.3.1.33 #de£ne 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

De£nition at line 837 of £le midas.h.

2.3.1.34 #de£ne LAM_SOURCE_CRATE(c) (c>>24)

Convert the coded LAM crate to Crate number.

Parameters:

- ***c*** coded crate

De£nition at line 849 of £le midas.h.

Referenced by poll_event().

2.3.1.35 #de£ne LAM_SOURCE_STATION(s) ((s) & 0xFFFFFF)

Convert the coded LAM station to Station number.

Parameters:

- ***s*** Slot number

De£nition at line 855 of £le midas.h.

Referenced by poll_event().

2.3.1.36 #de£ne LAM_STATION(s) (1<<(s-1))

Code the Station number bitwise for the LAM source.

Parameters:

s Slot number

De£nition at line 843 of £le midas.h.

2.3.1.37 #de£ne MAX_CLIENTS 64

client processes per buf/db

De£nition at line 669 of £le midas.h.

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

2.3.1.38 #de£ne MAX_EVENT_REQUESTS 10

event requests per client

De£nition at line 670 of £le midas.h.

Referenced by bm_remove_event_request().

2.3.1.39 #de£ne MAX_EVENT_SIZE 0x80000

maximum event size 512k

De£nition at line 663 of £le midas.h.

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

2.3.1.40 #de£ne MAX_EXPERIMENT 32

number of different exp.

De£nition at line 673 of £le midas.h.

Referenced by cm_connect_experiment1(), and cm_list_experiments().

2.3.1.41 #de£ne MAX_ODB_PATH 256

length of path in ODB

De£nition at line 672 of £le midas.h.

2.3.1.42 #de£ne MAX_OPEN_RECORDS 256

number of open DB records

De£nition at line 671 of £le midas.h.

2.3.1.43 #de£ne MIDAS_TCP_PORT 1175

De£nition at line 678 of £le midas.h.

Referenced by cm_list_experiments(), and cm_transition().

2.3.1.44 #de£ne MIDAS_VERSION "1.9.5"

De£nition at line 492 of £le midas.h.

2.3.1.45 #de£ne NAME_LENGTH 32

length of names, mult.of 8!

De£nition at line 667 of £le midas.h.

Referenced by cm_connect_experiment1(), and cm_set_client_info().

2.3.1.46 #de£ne NET_TCP_SIZE 0xFFFF

maximum TCP transfer

De£nition at line 657 of £le midas.h.

Referenced by rpc_send_event(), and scheduler().

2.3.1.47 #de£ne NET_UDP_SIZE 8192

maximum UDP transfer

De£nition at line 659 of £le midas.h.

2.3.1.48 #de£ne OPT_TCP_SIZE 8192

optimal TCP buffer size

De£nition at line 658 of £le midas.h.

2.3.1.49 #de£ne STRING_BANKLIST_MAX BANKLIST_MAX ∗ 4

for bk_list()

De£nition at line 675 of £le midas.h.

2.3.1.50 #de£ne TAPE_BUFFER_SIZE 0x8000

buffer size for taping data

De£nition at line 655 of £le midas.h.

2.3.1.51 #de£ne WATCHDOG_INTERVAL 1000

De£nition at line 683 of £le midas.h.

Referenced by cm_set_client_info(), cm_set_watchdog_params(), and ss_sleep().

2.3.2 Variable Documentation

2.3.2.1 INT _call_watchdog = TRUE [static]

De£nition at line 1746 of £le midas.c.

Referenced by cm_set_watchdog_params().

2.3.2.2 char _client_name[NAME_LENGTH] [static]

De£nition at line 1744 of £le midas.c.

2.3.2.3 HNDLE _hDB = 0 [static]

De£nition at line 1743 of £le midas.c.

Referenced by cm_set_experiment_database().

2.3.2.4 HNDLE _hKeyClient = 0 [static]

dox***

De£nition at line 1742 of £le midas.c.

Referenced by cm_set_experiment_database().

2.3.2.5 INT _mutex_alarm

De£nition at line 1748 of £le midas.c.

Referenced by db_close_database().

2.3.2.6 INT _mutex_elog

De£nition at line 1748 of £le midas.c.

Referenced by db_close_database().

2.3.2.7 char _path_name[MAX_STRING_LENGTH] [static]

De£nition at line 1745 of £le midas.c.

Referenced by cm_get_path(), and cm_set_path().

2.3.2.8 INT _watchdog_timeout = DEFAULT_WATCHDOG_TIMEOUT [static]

De£nition at line 1747 of £le midas.c.

Referenced by cm_set_watchdog_params().

2.4 Midas De£ne

De£nes

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

- #de£ne TID_KEY 15
- #de£ne TID_LINK 16
- #de£ne TID_LAST 17
- #de£ne SYNC 0
- #de£ne MODE_READ (1<<0)
- #de£ne RPC_OTIMEOUT 1
- #de£ne WF_WATCH_ME (1<<0)
- #de£ne TR_START (1<<0)
- #de£ne TR_STOP (1<<1)
- #de£ne TR_PAUSE (1<<2)
- #de£ne TR_RESUME (1<<3)
- #de£ne EQ_PERIODIC (1<<0)
- #de£ne EQ_POLLED (1<<1)
- #de£ne EQ_INTERRUPT (1<<2)
- #de£ne EQ_SLOW (1<<3)
- #de£ne EQ_MANUAL_TRIG (1<<4)
- #de£ne EQ_FRAGMENTED (1<<5)
- #de£ne EQ_EB (1<<6)
- #de£ne RO_RUNNING (1<<0)
- #de£ne RO_STOPPED (1<<1)
- #de£ne RO_PAUSED (1<<2)
- #de£ne RO_BOR (1<<3)
- #de£ne RO_EOR (1<<4)
- #de£ne RO_PAUSE (1<<5)
- #de£ne RO_RESUME (1<<6)
- #de£ne RO_TRANSITIONS (RO_BOR|RO_EOR|RO_PAUSE|RO_RESUME)
- #de£ne RO_ALWAYS (0xFF)
- #de£ne RO_ODB (1<<8)
- #de£ne MT_ERROR (1<<0)
- #de£ne MT_INFO (1<<1)
- #de£ne MT_DEBUG (1<<2)
- #de£ne MT_USER (1<<3)
- #de£ne MT_LOG (1<<4)
- #de£ne MT_TALK (1<<5)
- #de£ne MT_CALL (1<<6)
- #de£ne MT_ALL 0xFF
- #de£ne MERROR MT_ERROR, __FILE__, __LINE__
- #de£ne MINFO MT_INFO, __FILE__, __LINE__
- #de£ne MDEBUG MT_DEBUG, __FILE__, __LINE__
- #de£ne MUSER MT_USER, __FILE__, __LINE__
- #de£ne MLOG MT_LOG, __FILE__, __LINE__
- #de£ne MTALK MT_TALK, __FILE__, __LINE__
- #de£ne MCALL MT_CALL, __FILE__, __LINE__

2.4.1 De£ne Documentation

2.4.1.1 #de£ne ASYNC 1

De£nition at line 742 of £le midas.h.

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

2.4.1.2 #de£ne EQ_EB (1<<6)

Event run through the event builder

De£nition at line 791 of £le midas.h.

2.4.1.3 #de£ne EQ_FRAGMENTED (1<<5)

Fragmented Event

De£nition at line 790 of £le midas.h.

2.4.1.4 #de£ne EQ_INTERRUPT (1<<2)

Interrupt Event

De£nition at line 787 of £le midas.h.

2.4.1.5 #de£ne EQ_MANUAL_TRIG (1<<4)

Manual triggered Event

De£nition at line 789 of £le midas.h.

2.4.1.6 #de£ne EQ_PERIODIC (1<<0)

Periodic Event

De£nition at line 785 of £le midas.h.

2.4.1.7 #de£ne EQ_POLLED (1<<1)

Polling Event

De£nition at line 786 of £le midas.h.

2.4.1.8 #de£ne EQ_SLOW (1<<3)

Slow Control Event

De£nition at line 788 of £le midas.h.

Referenced by scheduler().

2.4.1.9 #de£ne EVENTID_ALL -1

De£nition at line 905 of £le midas.h.

Referenced by bm_match_event(), and cm_msg_register().

2.4.1.10 #de£ne FORMAT_ASCII 3

ASCII format

De£nition at line 706 of £le midas.h.

2.4.1.11 #de£ne FORMAT_DUMP 5

Dump (detailed ASCII) format

De£nition at line 708 of £le midas.h.

2.4.1.12 #de£ne FORMAT_FIXED 4

Fixed length binary records

De£nition at line 707 of £le midas.h.

2.4.1.13 #de£ne FORMAT_HBOOK 6

CERN hbook (rz) format

De£nition at line 709 of £le midas.h.

2.4.1.14 #de£ne FORMAT_MIDAS 1

MIDAS banks

De£nition at line 704 of £le midas.h.

Referenced by source_scan().

2.4.1.15 #de£ne FORMAT_ROOT 7

CERN ROOT format

De£nition at line 710 of £le midas.h.

2.4.1.16 #de£ne FORMAT_YBOS 2

YBOS banks

De£nition at line 705 of £le midas.h.

Referenced by source_scan().

2.4.1.17 #de£ne GET_ALL (1<<0)

get all events (consume)

De£nition at line 714 of £le midas.h.

Referenced by source_booking().

2.4.1.18 #de£ne GET_FARM (1<<2)

distribute events over several clients (farming)

De£nition at line 716 of £le midas.h.

2.4.1.19 #de£ne GET_SOME (1<<1)

get as much as possible (sampling)

De£nition at line 715 of £le midas.h.

Referenced by cm_msg_register().

2.4.1.20 #de£ne MCALL MT_CALL, __FILE__, __LINE__

info message for telephone call

De£nition at line 925 of £le midas.h.

2.4.1.21 #de£ne MDEBUG MT_DEBUG, __FILE__, __LINE__

-

De£nition at line 921 of £le midas.h.

Referenced by bm_¤ush_cache(), bm_push_event(), bm_receive_event(), bm_send_-event(), and cm_transition().

2.4.1.22 #de£ne MERROR MT_ERROR, __FILE__, __LINE__

-

De£nition at line 919 of £le midas.h.

Referenced by al_trigger_alarm(), analyzer_init(), bm_close_buffer(), bm_¤ush_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(), cm_check_client(), cm_check_deferred_transition(), cm_cleanup(), cm_connect_experiment1(), 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_create_key(), db_create_link(), db_create_record(), db_delete_key1(), db_enum_key(), db_£nd_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_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(), interrupt_routine(), load_fragment(), main(), register_equipment(), rpc_¤ush_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().

2.4.1.23 #de£ne MINFO MT_INFO, __FILE__, __LINE__

•

De£nition at line 920 of £le midas.h.

Referenced by bk_list(), 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().

2.4.1.24 #de£ne MLOG MT_LOG, __FILE__, __LINE__

info message which is only logged

De£nition at line 923 of £le midas.h.

2.4.1.25 #de£ne MODE_ALLOC (1<<7)

De£nition at line 750 of £le midas.h.

Referenced by db_open_record().

2.4.1.26 #de£ne MODE_DELETE (1<<2)

De£nition at line 748 of £le midas.h.

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

2.4.1.27 #de£ne MODE_EXCLUSIVE (1<<3)

De£nition at line 749 of £le midas.h.

Referenced by cm_cleanup(), and db_open_database().

2.4.1.28 #de£ne MODE_READ (1<<0)

Access modes

De£nition at line 746 of £le midas.h.

Referenced by analyzer_init(), 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_key(), db_open_-database(), db_open_record(), and register_equipment().

2.4.1.29 #de£ne MODE_WRITE (1<<1)

De£nition at line 747 of £le midas.h.

Referenced by cm_cleanup(), 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_key(), db_open_-database(), and register_equipment().

2.4.1.30 #de£ne MT_ALL 0xFF

-

De£nition at line 917 of £le midas.h.

Referenced by cm_connect_experiment1(), and main().

2.4.1.31 #de£ne MT_CALL (1<<6)

-

De£nition at line 916 of £le midas.h.

2.4.1.32 #de£ne MT_DEBUG (1<<2)

•

De£nition at line 912 of £le midas.h.

2.4.1.33 #de£ne MT_ERROR (1<<0)

•

De£nition at line 910 of £le midas.h.

2.4.1.34 #de£ne MT_INFO (1<<1)

•

De£nition at line 911 of £le midas.h.

2.4.1.35 #de£ne MT_LOG (1<<4)

•

De£nition at line 914 of £le midas.h.

2.4.1.36 #de£ne MT_TALK (1<<5)

•

De£nition at line 915 of £le midas.h.

2.4.1.37 #de£ne MT_USER (1<<3)

•

De£nition at line 913 of £le midas.h.

2.4.1.38 #de£ne MTALK MT_TALK, __FILE__, __LINE__

info message for speech system

De£nition at line 924 of £le midas.h.

Referenced by scan_fragment(), and scheduler().

2.4.1.39 #de£ne MUSER MT_USER, __FILE__, __LINE__

produced by interactive user

De£nition at line 922 of £le midas.h.

2.4.1.40 #de£ne RO_ALWAYS (0xFF)

Always (independent of the run status)

De£nition at line 805 of £le midas.h.

2.4.1.41 #de£ne RO_BOR (1<<3)

At the Begin of run

De£nition at line 799 of £le midas.h.

2.4.1.42 #de£ne RO_EOR (1<<4)

At the End of run

De£nition at line 800 of £le midas.h.

2.4.1.43 #de£ne RO_ODB (1<<8)

Submit data to ODB only

De£nition at line 807 of £le midas.h.

Referenced by interrupt_routine(), and scheduler().

2.4.1.44 #de£ne RO_PAUSE (1<<5)

Before pausing the run

De£nition at line 801 of £le midas.h.

2.4.1.45 #de£ne RO_PAUSED (1<<2)

???

De£nition at line 798 of £le midas.h.

2.4.1.46 #de£ne RO_RESUME (1<<6)

Before resuming the run

De£nition at line 802 of £le midas.h.

2.4.1.47 #de£ne RO_RUNNING (1<<0)

While running

De£nition at line 796 of £le midas.h.

2.4.1.48 #de£ne RO_STOPPED (1<<1)

Before stopping the run

De£nition at line 797 of £le midas.h.

2.4.1.49 #de£ne RO_TRANSITIONS (RO_BOR|RO_EOR|RO_PAUSE|RO_-RESUME)

At all transitions

De£nition at line 804 of £le midas.h.

2.4.1.50 #de£ne RPC_CLIENT_HANDLE 9

De£nition at line 762 of £le midas.h.

Referenced by cm_get_experiment_database(), and cm_set_client_info().

2.4.1.51 #de£ne RPC_CONVERT_FLAGS 7

De£nition at line 760 of £le midas.h.

Referenced by bm_receive_event(), db_get_record(), db_set_record(), and db_-update_record().

2.4.1.52 #de£ne RPC_FTCP 1

De£nition at line 768 of £le midas.h.

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

2.4.1.53 #de£ne RPC_NODELAY 12

De£nition at line 765 of £le midas.h.

Referenced by rpc_set_option().

2.4.1.54 #de£ne RPC_OCONVERT_FLAG 3

De£nition at line 756 of £le midas.h.

2.4.1.55 #de£ne RPC_ODB_HANDLE 8

De£nition at line 761 of £le midas.h.

Referenced by cm_get_experiment_database(), and cm_set_client_info().

2.4.1.56 #de£ne RPC_OHW_TYPE 4

De£nition at line 757 of £le midas.h.

Referenced by cm_connect_experiment1().

2.4.1.57 #de£ne RPC_OSERVER_NAME 6

De£nition at line 759 of £le midas.h.

2.4.1.58 #de£ne RPC_OSERVER_TYPE 5

De£nition at line 758 of £le 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().

2.4.1.59 #de£ne RPC_OTIMEOUT 1

RPC options

De£nition at line 754 of £le midas.h.

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

2.4.1.60 #de£ne RPC_OTRANSPORT 2

De£nition at line 755 of £le midas.h.

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

2.4.1.61 #de£ne RPC_SEND_SOCK 10

De£nition at line 763 of £le midas.h.

2.4.1.62 #de£ne RPC_TCP 0

De£nition at line 767 of £le midas.h.

Referenced by cm_transition(), db_send_changed_records(), scheduler(), and update_-odb().

2.4.1.63 #de£ne RPC_WATCHDOG_TIMEOUT 11

De£nition at line 764 of £le midas.h.

Referenced by cm_set_watchdog_params().

2.4.1.64 #de£ne STATE_PAUSED 2

MIDAS run paused

De£nition at line 699 of £le midas.h.

Referenced by scan_fragment(), and scheduler().

2.4.1.65 #de£ne STATE_RUNNING 3

MIDAS run running

De£nition at line 700 of £le midas.h.

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

2.4.1.66 #de£ne STATE_STOPPED 1

MIDAS run stopped

De£nition at line 698 of £le midas.h.

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

2.4.1.67 #de£ne SYNC 0

Synchronous / Asynchronous ¤ags

De£nition at line 741 of £le midas.h.

Referenced by close_buffers(), cm_check_deferred_transition(), cm_msg(), cm_-msg1(), interrupt_routine(), scheduler(), send_event(), source_scan(), and tr_stop().

2.4.1.68 #de£ne TID_ARRAY 13

array with unknown contents

De£nition at line 733 of £le midas.h.

2.4.1.69 #de£ne TID_BITFIELD 11

32 Bits Bit£eld 0 111... (32)

De£nition at line 731 of £le midas.h.

Referenced by db_sprintf().

2.4.1.70 #de£ne TID_BOOL 8

four bytes bool 0 1

De£nition at line 728 of £le midas.h.

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

2.4.1.71 #de£ne TID_BYTE 1

unsigned byte 0 255

De£nition at line 721 of £le midas.h.

Referenced by db_sprintf().

2.4.1.72 #de£ne TID_CHAR 3

single character 0 255

De£nition at line 723 of £le midas.h.

Referenced by db_sprintf().

2.4.1.73 #de£ne TID_DOUBLE 10

8 Byte ¤oat format

De£nition at line 730 of £le midas.h.

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

2.4.1.74 #de£ne TID_DWORD 6

four bytes 0 2^32-1

De£nition at line 726 of £le midas.h.

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

2.4.1.75 #de£ne TID_FLOAT 9

4 Byte ¤oat format

De£nition at line 729 of £le midas.h.

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

2.4.1.76 #de£ne TID_INT 7

signed dword -2^31 2^31-1

De£nition at line 727 of £le midas.h.

Referenced by 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(), register_equipment(), scheduler(), and tr_start().

2.4.1.77 #de£ne TID_KEY 15

key in online database

De£nition at line 735 of £le midas.h.

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

2.4.1.78 #de£ne TID_LAST 17

end of TID list indicator

De£nition at line 737 of £le midas.h.

2.4.1.79 #de£ne TID_LINK 16

link in online database

De£nition at line 736 of £le midas.h.

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

2.4.1.80 #de£ne TID_SBYTE 2

signed byte -128 127

De£nition at line 722 of £le midas.h.

Referenced by db_sprintf().

2.4.1.81 #de£ne TID_SHORT 5

signed word -32768 32767

De£nition at line 725 of £le midas.h.

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

2.4.1.82 #de£ne TID_STRING 12

zero terminated string

De£nition at line 732 of £le 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_save_xml_key(), db_set_data_index(), db_set_value(), db_sprintf(), el_submit(), load_fragment(), logger_root(), tr_start(), and update_odb().

2.4.1.83 #de£ne TID_STRUCT 14

structure with £xed length

De£nition at line 734 of £le midas.h.

Referenced by adc_summing(), and bk_close().

2.4.1.84 #de£ne TID_WORD 4

two bytes 0 65535

De£nition at line 724 of £le midas.h.

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

2.4.1.85 #de£ne TR_DEFERRED (1<<12)

De£nition at line 781 of £le midas.h.

Referenced by cm_check_deferred_transition().

2.4.1.86 #de£ne TR_PAUSE (1<<2)

Pause transition

De£nition at line 779 of £le midas.h.

Referenced by cm_register_transition(), cm_set_transition_sequence(), cm_-transition(), main(), send_all_periodic_events(), and tr_pause().

2.4.1.87 #de£ne TR_RESUME (1<<3)

Resume transition

De£nition at line 780 of £le midas.h.

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

2.4.1.88 #de£ne TR_START (1<<0)

Start transition

De£nition at line 777 of £le midas.h.

Referenced by cm_register_transition(), cm_set_transition_sequence(), cm_-transition(), main(), scheduler(), send_all_periodic_events(), and tr_start().

2.4.1.89 #de£ne TR_STOP (1<<1)

Stop transition

De£nition at line 778 of £le midas.h.

Referenced by cm_register_transition(), cm_set_transition_sequence(), cm_-transition(), main(), scan_fragment(), scheduler(), send_all_periodic_events(), and tr_stop().

2.4.1.90 #de£ne TRIGGER_ALL -1

De£nition at line 906 of £le midas.h.

Referenced by bm_match_event(), and cm_msg_register().

2.4.1.91 #de£ne WF_CALL_WD (1<<1)

De£nition at line 773 of £le midas.h.

2.4.1.92 #de£ne WF_WATCH_ME (1<<0)

Watchdog ¤ags

De£nition at line 772 of £le midas.h.

2.5 Midas Macros

De£nes

- #de£ne MAX(a, b) (((a) > (b)) ? (a) : (b))
- #de£ne MIN(a, b) (((a) < (b)) ? (a) : (b))

- #de£ne ALIGN8(x) (((x)+7) & ∼7)
- #de£ne VALIGN(adr, align) (((PTYPE) (adr)+align-1) & ∼(align-1))

2.5.1 De£ne Documentation

2.5.1.1 #de£ne ALIGN8(x) (((x)+7) & ∼7)

Align macro for data alignment on 8-byte boundary

De£nition at line 890 of £le midas.h.

Referenced by bk_close(), bk_£nd(), bk_iterate(), bk_locate(), bk_swap(), bm_-¤ush_cache(), bm_push_event(), bm_receive_event(), bm_send_event(), db_open_-database(), and rpc_send_event().

2.5.1.2 #de£ne MAX(a, b) (((a) > (b)) ? (a) : (b))

MAX

De£nition at line 877 of £le midas.h.

Referenced by cm_execute(), and scheduler().

2.5.1.3 #de£ne MIN(a, b) (((a) < (b)) ? (a) : (b))

MIN

De£nition at line 883 of £le midas.h.

Referenced by update_odb().

2.5.1.4 #de£ne VALIGN(adr, align) (((PTYPE) (adr)+align-1) & ∼(align-1))

Align macro for variable data alignment

De£nition at line 894 of £le midas.h.

Referenced by db_get_record_size(), and update_odb().

2.6 Midas Error de£nition

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

2.7 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

2.8 Status and error codes

De£nes

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

2.8.1 De£ne Documentation

2.8.1.1 #de£ne CM_DB_ERROR 104

db access error

De£nition at line 945 of £le midas.h.

2.8.1.2 #de£ne CM_DEFERRED_TRANSITION 110

-

De£nition at line 951 of £le midas.h.

2.8.1.3 #de£ne CM_INVALID_TRANSITION 113

-

De£nition at line 954 of £le midas.h.

2.8.1.4 #de£ne CM_NO_CLIENT 103

nobody

De£nition at line 944 of £le midas.h.

2.8.1.5 #de£ne CM_SET_ERROR 102

set

De£nition at line 943 of £le midas.h.

2.8.1.6 #de£ne CM_SHUTDOWN 107

-

De£nition at line 948 of £le midas.h.

2.8.1.7 #de£ne CM_SUCCESS 1

Same

De£nition at line 942 of £le midas.h.

Referenced by main().

2.8.1.8 #de£ne CM_TIMEOUT 112

•

De£nition at line 953 of £le midas.h.

2.8.1.9 #de£ne CM_TOO_MANY_REQUESTS 114

•

De£nition at line 955 of £le midas.h.

2.8.1.10 #de£ne CM_TRANSITION_IN_PROGRESS 111

•

De£nition at line 952 of £le midas.h.

2.8.1.11 #de£ne CM_UNDEF_ENVIRON 109

•

De£nition at line 950 of £le midas.h.

2.8.1.12 #de£ne CM_UNDEF_EXP 105

•

De£nition at line 946 of £le midas.h.

2.8.1.13 #de£ne CM_VERSION_MISMATCH 106

•

De£nition at line 947 of £le midas.h.

2.8.1.14 #de£ne CM_WRONG_PASSWORD 108

•

De£nition at line 949 of £le midas.h.

2.8.1.15 #de£ne SUCCESS 1

Success

De£nition at line 941 of £le midas.h.

2.9 Buffer Manager error codes

De£nes

- #de£ne BM_SUCCESS 1
- #de£ne BM_CREATED 202
- #de£ne BM_NO_MEMORY 203
- #de£ne BM_INVALID_NAME 204
- #de£ne BM_INVALID_HANDLE 205
- #de£ne BM_NO_SLOT 206
- #de£ne BM_NO_MUTEX 207
- #de£ne BM_NOT_FOUND 208
- #de£ne BM_ASYNC_RETURN 209
- #de£ne BM_TRUNCATED 210
- #de£ne BM_MULTIPLE_HOSTS 211
- #de£ne BM_MEMSIZE_MISMATCH 212
- #de£ne BM_CONFLICT 213
- #de£ne BM_EXIT 214
- #de£ne BM_INVALID_PARAM 215
- #de£ne BM_MORE_EVENTS 216
- #de£ne BM_INVALID_MIXING 217
- #de£ne BM_NO_SHM 218

2.9.1 De£ne Documentation

2.9.1.1 #de£ne BM_ASYNC_RETURN 209

-

De£nition at line 971 of £le midas.h.

Referenced by scan_fragment(), and source_scan().

2.9.1.2 #de£ne BM_CONFLICT 213

•

De£nition at line 975 of £le midas.h.

2.9.1.3 #de£ne BM_CREATED 202

•

De£nition at line 964 of £le midas.h.

2.9.1.4 #de£ne BM_EXIT 214

•

De£nition at line 976 of £le midas.h.

2.9.1.5 #de£ne BM_INVALID_HANDLE 205

•

De£nition at line 967 of £le midas.h.

2.9.1.6 #de£ne BM_INVALID_MIXING 217

•

De£nition at line 979 of £le midas.h.

2.9.1.7 #de£ne BM_INVALID_NAME 204

•

De£nition at line 966 of £le midas.h.

2.9.1.8 #de£ne BM_INVALID_PARAM 215

•

De£nition at line 977 of £le midas.h.

2.9.1.9 #de£ne BM_MEMSIZE_MISMATCH 212

•

De£nition at line 974 of £le midas.h.

2.9.1.10 #de£ne BM_MORE_EVENTS 216

•

De£nition at line 978 of £le midas.h.

2.9.1.11 #de£ne BM_MULTIPLE_HOSTS 211

•

De£nition at line 973 of £le midas.h.

2.9.1.12 #de£ne BM_NO_MEMORY 203

•

De£nition at line 965 of £le midas.h.

2.9.1.13 #de£ne BM_NO_MUTEX 207

•

De£nition at line 969 of £le midas.h.

2.9.1.14 #de£ne BM_NO_SHM 218

•

De£nition at line 980 of £le midas.h.

2.9.1.15 #de£ne BM_NO_SLOT 206

•

De£nition at line 968 of £le midas.h.

2.9.1.16 #de£ne BM_NOT_FOUND 208

•

De£nition at line 970 of £le midas.h.

2.9.1.17 #de£ne BM_SUCCESS 1

•

De£nition at line 963 of £le midas.h.

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

2.9.1.18 #de£ne BM_TRUNCATED 210

•

De£nition at line 972 of £le midas.h.

2.10 Online Database error codes

De£nes

- #de£ne DB_SUCCESS 1
- #de£ne DB_CREATED 302
- #de£ne DB_NO_MEMORY 303
- #de£ne DB_INVALID_NAME 304
- #de£ne DB_INVALID_HANDLE 305
- #de£ne DB_NO_SLOT 306
- #de£ne DB_NO_MUTEX 307
- #de£ne DB_MEMSIZE_MISMATCH 308
- #de£ne DB_INVALID_PARAM 309
- #de£ne DB_FULL 310
- #de£ne DB_KEY_EXIST 311
- #de£ne DB_NO_KEY 312
- #de£ne DB_KEY_CREATED 313
- #de£ne DB_TRUNCATED 314
- #de£ne DB_TYPE_MISMATCH 315
- #de£ne DB_NO_MORE_SUBKEYS 316

- #de£ne DB_FILE_ERROR 317
- #de£ne DB_NO_ACCESS 318
- #de£ne DB_STRUCT_SIZE_MISMATCH 319
- #de£ne DB_OPEN_RECORD 320
- #de£ne DB_OUT_OF_RANGE 321
- #de£ne DB_INVALID_LINK 322
- #de£ne DB_CORRUPTED 323
- #de£ne DB_STRUCT_MISMATCH 324
- #de£ne DB_TIMEOUT 325
- #de£ne DB_VERSION_MISMATCH 326

2.10.1 De£ne Documentation

2.10.1.1 #de£ne DB_CORRUPTED 323

-

De£nition at line 1009 of £le midas.h.

2.10.1.2 #de£ne DB_CREATED 302

-

De£nition at line 988 of £le midas.h.

2.10.1.3 #de£ne DB_FILE_ERROR 317

-

De£nition at line 1003 of £le midas.h.

2.10.1.4 #de£ne DB_FULL 310

-

De£nition at line 996 of £le midas.h.

2.10.1.5 #de£ne DB_INVALID_HANDLE 305

•

De£nition at line 991 of £le midas.h.

2.10.1.6 #de£ne DB_INVALID_LINK 322

•

De£nition at line 1008 of £le midas.h.

2.10.1.7 #de£ne DB_INVALID_NAME 304

•

De£nition at line 990 of £le midas.h.

2.10.1.8 #de£ne DB_INVALID_PARAM 309

•

De£nition at line 995 of £le midas.h.

2.10.1.9 #de£ne DB_KEY_CREATED 313

•

De£nition at line 999 of £le midas.h.

2.10.1.10 #de£ne DB_KEY_EXIST 311

•

De£nition at line 997 of £le midas.h.

2.10.1.11 #de£ne DB_MEMSIZE_MISMATCH 308

•

De£nition at line 994 of £le midas.h.

2.10.1.12 #de£ne DB_NO_ACCESS 318

-

De£nition at line 1004 of £le midas.h.

2.10.1.13 #de£ne DB_NO_KEY 312

-

De£nition at line 998 of £le midas.h.

2.10.1.14 #de£ne DB_NO_MEMORY 303

-

De£nition at line 989 of £le midas.h.

2.10.1.15 #de£ne DB_NO_MORE_SUBKEYS 316

-

De£nition at line 1002 of £le midas.h.

2.10.1.16 #de£ne DB_NO_MUTEX 307

-

De£nition at line 993 of £le midas.h.

2.10.1.17 #de£ne DB_NO_SLOT 306

-

De£nition at line 992 of £le midas.h.

2.10.1.18 #de£ne DB_OPEN_RECORD 320

-

De£nition at line 1006 of £le midas.h.

2.10.1.19 #de£ne DB_OUT_OF_RANGE 321

•

De£nition at line 1007 of £le midas.h.

2.10.1.20 #de£ne DB_STRUCT_MISMATCH 324

•

De£nition at line 1010 of £le midas.h.

2.10.1.21 #de£ne DB_STRUCT_SIZE_MISMATCH 319

•

De£nition at line 1005 of £le midas.h.

2.10.1.22 #de£ne DB_SUCCESS 1

•

De£nition at line 987 of £le midas.h.

Referenced by al_trigger_alarm(), cm_connect_experiment1(), cm_shutdown(), cm_-transition(), db_check_record(), and db_delete_key1().

2.10.1.23 #de£ne DB_TIMEOUT 325

•

De£nition at line 1011 of £le midas.h.

2.10.1.24 #de£ne DB_TRUNCATED 314

•

De£nition at line 1000 of £le midas.h.

2.10.1.25 #de£ne DB_TYPE_MISMATCH 315

•

De£nition at line 1001 of £le midas.h.

2.10.1.26 #de£ne DB_VERSION_MISMATCH 326

•

De£nition at line 1012 of £le midas.h.

2.11 System Services error code

De£nes

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

2.11.1 De£ne Documentation

2.11.1.1 #de£ne SS_ABORT 415

•

De£nition at line 1033 of £le midas.h.

2.11.1.2 #de£ne SS_CLIENT_RECV 414

•

De£nition at line 1032 of £le midas.h.

2.11.1.3 #de£ne SS_CREATED 402

•

De£nition at line 1020 of £le midas.h.

Referenced by bm_open_buffer(), cm_connect_experiment1(), and dm_buffer_-create().

2.11.1.4 #de£ne SS_DEV_BUSY 418

•

De£nition at line 1036 of £le midas.h.

2.11.1.5 #de£ne SS_END_OF_FILE 423

•

De£nition at line 1041 of £le midas.h.

2.11.1.6 #de£ne SS_END_OF_TAPE 422

•

De£nition at line 1040 of £le midas.h.

2.11.1.7 #de£ne SS_EXIT 416

•

De£nition at line 1034 of £le midas.h.

2.11.1.8 #de£ne SS_FILE_ERROR 407

•

De£nition at line 1025 of £le midas.h.

2.11.1.9 #de£ne SS_FILE_EXISTS 424

•

De£nition at line 1042 of £le midas.h.

2.11.1.10 #de£ne SS_INVALID_ADDRESS 406

•

De£nition at line 1024 of £le midas.h.

2.11.1.11 #de£ne SS_INVALID_FORMAT 426

•

De£nition at line 1044 of £le midas.h.

2.11.1.12 #de£ne SS_INVALID_HANDLE 405

•

De£nition at line 1023 of £le midas.h.

2.11.1.13 #de£ne SS_INVALID_NAME 404

•

De£nition at line 1022 of £le midas.h.

2.11.1.14 #de£ne SS_IO_ERROR 419

•

De£nition at line 1037 of £le midas.h.

2.11.1.15 #de£ne SS_NO_DRIVER 421

•

De£nition at line 1039 of £le midas.h.

2.11.1.16 #de£ne SS_NO_MEMORY 403

•

De£nition at line 1021 of £le midas.h.

Referenced by bm_open_buffer(), and db_open_database().

2.11.1.17 #de£ne SS_NO_MUTEX 408

•

De£nition at line 1026 of £le midas.h.

2.11.1.18 #de£ne SS_NO_PROCESS 409

•

De£nition at line 1027 of £le midas.h.

2.11.1.19 #de£ne SS_NO_ROOT 427

•

De£nition at line 1045 of £le midas.h.

2.11.1.20 #de£ne SS_NO_SPACE 425

•

De£nition at line 1043 of £le midas.h.

2.11.1.21 #de£ne SS_NO_TAPE 417

•

De£nition at line 1035 of £le midas.h.

2.11.1.22 #de£ne SS_NO_THREAD 410

•

De£nition at line 1028 of £le midas.h.

2.11.1.23 #de£ne SS_SERVER_RECV 413

•

De£nition at line 1031 of £le midas.h.

2.11.1.24 #de£ne SS_SOCKET_ERROR 411

•

De£nition at line 1029 of £le midas.h.

2.11.1.25 #de£ne SS_SUCCESS 1

•

De£nition at line 1019 of £le midas.h.

Referenced by db_open_database(), and ss_thread_kill().

2.11.1.26 #de£ne SS_TAPE_ERROR 420

•

De£nition at line 1038 of £le midas.h.

2.11.1.27 #de£ne SS_TIMEOUT 412

•

De£nition at line 1030 of £le midas.h.

2.12 Remote Procedure Calls error codes

De£nes

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

2.12.1 De£ne Documentation

2.12.1.1 #de£ne RPC_ABORT SS_ABORT

•

De£nition at line 1053 of £le midas.h.

2.12.1.2 #de£ne RPC_CONNCLOSED 507

•

De£nition at line 1059 of £le midas.h.

2.12.1.3 #de£ne RPC_DOUBLE_DEFINED 511

•

De£nition at line 1063 of £le midas.h.

2.12.1.4 #de£ne RPC_EXCEED_BUFFER 505

•

De£nition at line 1057 of £le midas.h.

2.12.1.5 #de£ne RPC_INVALID_ID 508

•

De£nition at line 1060 of £le midas.h.

2.12.1.6 #de£ne RPC_NET_ERROR 503

•

De£nition at line 1055 of £le midas.h.

2.12.1.7 #de£ne RPC_NO_CONNECTION 502

•

De£nition at line 1054 of £le midas.h.

2.12.1.8 #de£ne RPC_NO_MEMORY 510

•

De£nition at line 1062 of £le midas.h.

2.12.1.9 #de£ne RPC_NOT_REGISTERED 506

•

De£nition at line 1058 of £le midas.h.

2.12.1.10 #de£ne RPC_SHUTDOWN 509

•

De£nition at line 1061 of £le midas.h.

Referenced by scan_fragment(), and scheduler().

2.12.1.11 #de£ne RPC_SUCCESS 1

•

De£nition at line 1052 of £le midas.h.

2.12.1.12 #de£ne RPC_TIMEOUT 504

•

De£nition at line 1056 of £le midas.h.

2.13 Other errors

De£nes

- #de£ne FE_SUCCESS 1
- #de£ne FE_ERR_ODB 602
- #de£ne FE_ERR_HW 603
- #de£ne FE_ERR_DISABLED 604
- #de£ne FE_ERR_DRIVER 605
- #de£ne HS_SUCCESS 1
- #de£ne HS_FILE_ERROR 702
- #de£ne HS_NO_MEMORY 703
- #de£ne HS_TRUNCATED 704
- #de£ne HS_WRONG_INDEX 705
- #de£ne HS_UNDEFINED_EVENT 706

- #de£ne HS_UNDEFINED_VAR 707
- #de£ne FTP_SUCCESS 1
- #de£ne FTP_NET_ERROR 802
- #de£ne FTP_FILE_ERROR 803
- #de£ne FTP_RESPONSE_ERROR 804
- #de£ne FTP_INVALID_ARG 805
- #de£ne EL_SUCCESS 1
- #de£ne EL_FILE_ERROR 902
- #de£ne EL_NO_MESSAGE 903
- #de£ne EL_TRUNCATED 904
- #de£ne EL_FIRST_MSG 905
- #de£ne EL_LAST_MSG 906
- #de£ne AL_SUCCESS 1
- #de£ne AL_INVALID_NAME 1002
- #de£ne AL_ERROR_ODB 1003
- #de£ne AL_RESET 1004
- #de£ne CMD_INIT (1<<0)
- #de£ne CMD_WRITE 100
- #de£ne CMD_INTERRUPT_ENABLE 100
- #de£ne BD_GETS(s, z, p, t) info $\rightarrow$ bd(CMD_GETS, info $\rightarrow$ bd_info, s, z, p, t)

2.13.1 De£ne Documentation

2.13.1.1 #de£ne AL_ERROR_ODB 1003

•

De£nition at line 1107 of £le midas.h.

2.13.1.2 #de£ne AL_INVALID_NAME 1002

•

De£nition at line 1106 of £le midas.h.

2.13.1.3 #de£ne AL_RESET 1004

•

De£nition at line 1108 of £le midas.h.

2.13.1.4 #de£ne AL_SUCCESS 1

•

De£nition at line 1105 of £le midas.h.

2.13.1.5 #de£ne BD_GETS(s, z, p, t) info → bd(CMD_GETS, info → bd_info, s, z, p, t)

macros for bus driver access

De£nition at line 1148 of £le midas.h.

2.13.1.6 #de£ne BD_PUTS(s) info → bd(CMD_PUTS, info → bd_info, s)

De£nition at line 1150 of £le midas.h.

2.13.1.7 #de£ne BD_READS(s, z, p, t) info → bd(CMD_READ, info → bd_info, s, z, p, t)

De£nition at line 1149 of £le midas.h.

2.13.1.8 #de£ne BD_WRITES(s) info → bd(CMD_WRITE, info → bd_info, s)

De£nition at line 1151 of £le midas.h.

2.13.1.9 #de£ne CMD_DEBUG 104

De£nition at line 1136 of £le midas.h.

2.13.1.10 #de£ne CMD_DISABLE_COMMAND (1<<16)

De£nition at line 1128 of £le midas.h.

2.13.1.11 #de£ne CMD_ENABLE_COMMAND (1<<15)

De£nition at line 1127 of £le midas.h.

2.13.1.12 #de£ne CMD_EXIT (1<<1)

De£nition at line 1113 of £le midas.h.

Referenced by main().

2.13.1.13 #de£ne CMD_GET (1<<5)

De£nition at line 1117 of £le midas.h.

2.13.1.14 #de£ne CMD_GET_ALL (1<<6)

De£nition at line 1118 of £le midas.h.

2.13.1.15 #de£ne CMD_GET_CURRENT (1<<7)

De£nition at line 1119 of £le midas.h.

2.13.1.16 #de£ne CMD_GET_CURRENT_ALL (1<<8)

De£nition at line 1120 of £le midas.h.

2.13.1.17 #de£ne CMD_GET_DEFAULT_NAME (1<<12)

De£nition at line 1124 of £le midas.h.

2.13.1.18 #de£ne CMD_GET_DEFAULT_THRESHOLD (1<<13)

De£nition at line 1125 of £le midas.h.

2.13.1.19 #de£ne CMD_GET_DEMAND (1<<11)

De£nition at line 1123 of £le midas.h.

2.13.1.20 #de£ne CMD_GETS 103

De£nition at line 1135 of £le midas.h.

2.13.1.21 #de£ne CMD_IDLE (1<<2)

De£nition at line 1114 of £le midas.h.

Referenced by scheduler().

2.13.1.22 #de£ne CMD_INIT (1<<0)

Slow control commands error code

De£nition at line 1112 of £le midas.h.

Referenced by register_equipment().

2.13.1.23 #de£ne CMD_INTERRUPT_ATTACH 102

De£nition at line 1143 of £le midas.h.

Referenced by interrupt_con£gure(), and register_equipment().

2.13.1.24 #de£ne CMD_INTERRUPT_DETACH 103

De£nition at line 1144 of £le midas.h.

Referenced by interrupt_con£gure(), and main().

2.13.1.25 #de£ne CMD_INTERRUPT_DISABLE 101

De£nition at line 1142 of £le midas.h.

Referenced by interrupt_con£gure(), interrupt_enable(), and main().

2.13.1.26 #de£ne CMD_INTERRUPT_ENABLE 100

Commands for interrupt events error code

De£nition at line 1141 of £le midas.h.

Referenced by interrupt_con£gure(), and interrupt_enable().

2.13.1.27 #de£ne CMD_NAME 105

De£nition at line 1137 of £le midas.h.

2.13.1.28 #de£ne CMD_PUTS 102

De£nition at line 1134 of £le midas.h.

2.13.1.29 #de£ne CMD_READ 101

De£nition at line 1133 of £le midas.h.

2.13.1.30 #de£ne CMD_SET (1<<3)

De£nition at line 1115 of £le midas.h.

2.13.1.31 #de£ne CMD_SET_ALL (1<<4)

De£nition at line 1116 of £le midas.h.

2.13.1.32 #de£ne CMD_SET_CURRENT_LIMIT (1<<9)

De£nition at line 1121 of £le midas.h.

2.13.1.33 #de£ne CMD_SET_CURRENT_LIMIT_ALL (1<<10)

De£nition at line 1122 of £le midas.h.

2.13.1.34 #de£ne CMD_SET_LABEL (1<<14)

De£nition at line 1126 of £le midas.h.

2.13.1.35 #de£ne CMD_WRITE 100

Bus driver commands

De£nition at line 1132 of £le midas.h.

2.13.1.36 #de£ne EL_FILE_ERROR 902

•

De£nition at line 1097 of £le midas.h.

2.13.1.37 #de£ne EL_FIRST_MSG 905

•

De£nition at line 1100 of £le midas.h.

2.13.1.38 #de£ne EL_LAST_MSG 906

•

De£nition at line 1101 of £le midas.h.

2.13.1.39 #de£ne EL_NO_MESSAGE 903

•

De£nition at line 1098 of £le midas.h.

2.13.1.40 #de£ne EL_SUCCESS 1

•

De£nition at line 1096 of £le midas.h.

2.13.1.41 #de£ne EL_TRUNCATED 904

•

De£nition at line 1099 of £le midas.h.

2.13.1.42 #de£ne FE_ERR_DISABLED 604

•

De£nition at line 1073 of £le midas.h.

2.13.1.43 #de£ne FE_ERR_DRIVER 605

•

De£nition at line 1074 of £le midas.h.

2.13.1.44 #de£ne FE_ERR_HW 603

•

De£nition at line 1072 of £le midas.h.

2.13.1.45 #de£ne FE_ERR_ODB 602

•

De£nition at line 1071 of £le midas.h.

2.13.1.46 #de£ne FE_SUCCESS 1

•

De£nition at line 1070 of £le midas.h.

2.13.1.47 #de£ne FTP_FILE_ERROR 803

-

De£nition at line 1090 of £le midas.h.

2.13.1.48 #de£ne FTP_INVALID_ARG 805

-

De£nition at line 1092 of £le midas.h.

2.13.1.49 #de£ne FTP_NET_ERROR 802

-

De£nition at line 1089 of £le midas.h.

2.13.1.50 #de£ne FTP_RESPONSE_ERROR 804

-

De£nition at line 1091 of £le midas.h.

2.13.1.51 #de£ne FTP_SUCCESS 1

-

De£nition at line 1088 of £le midas.h.

2.13.1.52 #de£ne HS_FILE_ERROR 702

-

De£nition at line 1079 of £le midas.h.

2.13.1.53 #de£ne HS_NO_MEMORY 703

-

De£nition at line 1080 of £le midas.h.

2.13.1.54 #de£ne HS_SUCCESS 1

•

De£nition at line 1078 of £le midas.h.

2.13.1.55 #de£ne HS_TRUNCATED 704

•

De£nition at line 1081 of £le midas.h.

2.13.1.56 #de£ne HS_UNDEFINED_EVENT 706

•

De£nition at line 1083 of £le midas.h.

2.13.1.57 #de£ne HS_UNDEFINED_VAR 707

•

De£nition at line 1084 of £le midas.h.

2.13.1.58 #de£ne HS_WRONG_INDEX 705

•

De£nition at line 1082 of £le midas.h.

2.14 Buffer Section

Data Structures

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

De£nes

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

2.14.1 De£ne Documentation

2.14.1.1 #de£ne EVENT_ID(e) ((((EVENT_HEADER ∗) e)-1) → event_id)

EVENT_ID Extract or set the event ID £eld pointed by the argument..

Parameters:

e pointer to the midas event (pevent)

De£nition at line 1196 of £le midas.h.

2.14.1.2 #de£ne EVENT_SOURCE(e, o) (∗ (INT∗) (e+o))

De£nition at line 1211 of £le midas.h.

2.14.1.3 #de£ne EVENTID_BOR ((short int) 0x8000)

Begin-of-run

De£nition at line 1215 of £le midas.h.

2.14.1.4 #de£ne EVENTID_EOR ((short int) 0x8001)

End-of-run

De£nition at line 1216 of £le midas.h.

2.14.1.5 #de£ne EVENTID_FRAG ((unsigned short) 0xD000)

De£nition at line 1222 of £le midas.h.

2.14.1.6 #de£ne EVENTID_FRAG1 ((unsigned short) 0xC000)

fragmented events

De£nition at line 1221 of £le midas.h.

Referenced by bm_match_event(), and bm_push_event().

2.14.1.7 #de£ne EVENTID_MESSAGE ((short int) 0x8002)

Message events

De£nition at line 1217 of £le midas.h.

Referenced by cm_msg(), and cm_msg1().

2.14.1.8 #de£ne MIDAS_MAGIC 0x494d

'MI'

De£nition at line 1226 of £le midas.h.

2.14.1.9 #de£ne SERIAL_NUMBER(e) ((((EVENT_HEADER ∗) e)-1) → serial_number)

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

Parameters:

e pointer to the midas event (pevent)

De£nition at line 1203 of £le midas.h.

2.14.1.10 #de£ne TIME_STAMP(e) ((((EVENT_HEADER ∗) e)-1) → time_stamp)

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

Parameters:

e pointer to the midas event (pevent)

De£nition at line 1210 of £le midas.h.

2.14.1.11 #de£ne TRIGGER_MASK(e) ((((EVENT_HEADER ∗) e)-1) → trigger_mask)

TRIGGER_MASK Extract or set the trigger mask £eld pointed by the argument.

Parameters:

e pointer to the midas event (pevent)

De£nition at line 1189 of £le midas.h.

2.15 Equipment related

Data Structures

- struct BUS_DRIVER
- struct DEVICE_DRIVER
- struct EQUIPMENT_INFO
- struct EQUIPMENT_STATS
- struct eqpmnt

De£nes

- #de£ne DF_INPUT (1<<0)
- #de£ne DF_OUTPUT (1<<1)
- #de£ne DF_PRIO_DEVICE (1<<2)
- #de£ne DF_READ_ONLY (1<<3)

2.15.1 De£ne Documentation

2.15.1.1 #de£ne DF_INPUT (1<<0)

channel is input

De£nition at line 1329 of £le midas.h.

2.15.1.2 #de£ne DF_OUTPUT (1<<1)

channel is output

De£nition at line 1330 of £le midas.h.

2.15.1.3 #de£ne DF_PRIO_DEVICE (1<<2)

get demand values from device instead of ODB

De£nition at line 1331 of £le midas.h.

2.15.1.4 #de£ne DF_READ_ONLY (1<<3)

never write demand values to device

De£nition at line 1332 of £le midas.h.

2.15.2 Typedef Documentation

2.15.2.1 typedef struct eqpmnt EQUIPMENT

Referenced by close_buffers(), scan_fragment(), scheduler(), send_event(), and tr_-stop().

2.15.2.2 typedef struct eqpmnt* PEQUIPMENT

De£nition at line 1373 of £le midas.h.

2.16 Bank related

Data Structures

- struct BANK_HEADER
- struct BANK
- struct BANK32
- struct TAG
- struct BANK_LIST

De£nes

- #de£ne BANK_FORMAT_VERSION 1
- #de£ne BANK_FORMAT_32BIT (1<<4)

2.16.1 De£ne Documentation

2.16.1.1 #de£ne BANK_FORMAT_32BIT (1<<4)

•

De£nition at line 1408 of £le midas.h.

2.16.1.2 #de£ne BANK_FORMAT_VERSION 1

•

De£nition at line 1407 of £le midas.h.

Referenced by bk_init32().

2.17 Analyzer related

Data Structures

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

2.17.1 De£ne Documentation

2.17.1.1 #de£ne DEF_TEST(t) extern ANA_TEST t;

De£nition at line 1525 of £le midas.h.

2.17.1.2 #de£ne SET_TEST(t, v) { if (!t.registered) test_register(&t); t.value = (v); }

De£nition at line 1519 of £le midas.h.

Referenced by adc_summing().

2.17.1.3 #de£ne TEST(t) (t.value)

De£nition at line 1520 of £le midas.h.

2.18 History related

Data Structures

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

2.18.1 De£ne Documentation

2.18.1.1 #de£ne RT_DATA (∗((DWORD ∗) "HSDA"))

De£nition at line 1536 of £le midas.h.

2.18.1.2 #de£ne RT_DEF (∗((DWORD ∗) "HSDF"))

De£nition at line 1537 of £le midas.h.

2.19 ODB runinfo related

Data Structures

- struct RUNINFO

2.19.1 De£ne Documentation

2.19.1.1 #de£ne 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 }
```

De£nition at line 1594 of £le midas.h.

Referenced by analyzer_init(), and cm_connect_experiment1().

2.20 Alarm related

2.20.1 Detailed Description

Alarm structre.

Data Structures

- struct PROGRAM_INFO
- struct ALARM_CLASS
- struct ALARM

De£nes

- #de£ne AT_INTERNAL 1
- #de£ne AT_PROGRAM 2
- #de£ne AT_EVALUATED 3
- #de£ne AT_PERIODIC 4
- #de£ne AT_LAST 4

2.20.2 De£ne Documentation

2.20.2.1 #de£ne 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 }
```

De£nition at line 1666 of £le midas.h.

2.20.2.2 #de£ne 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 }
```

De£nition at line 1696 of £le midas.h.

Referenced by al_trigger_alarm().

2.20.2.3 #de£ne 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 }
```

De£nition at line 1711 of £le midas.h.

2.20.2.4 #de£ne AT_EVALUATED 3

-

De£nition at line 1633 of £le midas.h.

Referenced by al_trigger_alarm().

2.20.2.5 #de£ne AT_INTERNAL 1

•

De£nition at line 1631 of £le midas.h.

2.20.2.6 #de£ne AT_LAST 4

•

De£nition at line 1635 of £le midas.h.

2.20.2.7 #de£ne AT_PERIODIC 4

•

De£nition at line 1634 of £le midas.h.

2.20.2.8 #de£ne AT_PROGRAM 2

•

De£nition at line 1632 of £le midas.h.

2.20.2.9 #de£ne 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 }
```

De£nition at line 1637 of £le midas.h.

Referenced by cm_set_client_info().

2.21 The ybos.h & ybos.c

Modules

- group YBOS De£ne
- group YBOS error code
- group YBOS Macros
- group YBOS Bank Functions (ybk_xxx)

2.22 YBOS De£ne

De£nes

- #de£ne YBOS_PHYREC_SIZE 8192
- #de£ne YBOS_BUFFER_SIZE 3∗(YBOS_PHYREC_SIZE<<2) + MAX_-EVENT_SIZE + 128
- #de£ne YB_BANKLIST_MAX 32
- #de£ne YB_STRING_BANKLIST_MAX YB_BANKLIST_MAX ∗ 4
- #de£ne H_BLOCK_SIZE 0
- #de£ne H_BLOCK_NUM 1
- #de£ne H_HEAD_LEN 2
- #de£ne H_START 3
- #de£ne D_RECORD 1
- #de£ne D_HEADER 2
- #de£ne D_EVTLEN 3
- #de£ne YB_COMPLETE 1
- #de£ne YB_INCOMPLETE 2
- #de£ne YB_NO_RECOVER -1
- #de£ne YB_NO_RUN 0
- #de£ne YB_ADD_RUN 1
- #de£ne DSP_RAW 1
- #de£ne DSP_BANK 2
- #de£ne DSP_UNK 0
- #de£ne DSP_DEC 1
- #de£ne DSP_HEX 2
- #de£ne DSP_ASC 3
- #de£ne I2_BKTYPE 1
- #de£ne A1_BKTYPE 2
- #de£ne I4_BKTYPE 3
- #de£ne F4_BKTYPE 4
- #de£ne D8_BKTYPE 5
- #de£ne I1_BKTYPE 8
- #de£ne MAX_BKTYPE I1_BKTYPE+1

2.22.1 De£ne Documentation

2.22.1.1 #de£ne A1_BKTYPE 2

ASCII 1 byte

De£nition at line 342 of £le ybos.h.

2.22.1.2 #de£ne D8_BKTYPE 5

Double 8 bytes

De£nition at line 345 of £le ybos.h.

2.22.1.3 #de£ne D_EVTLEN 3

YBOS

De£nition at line 175 of £le ybos.h.

2.22.1.4 #de£ne D_HEADER 2

YBOS

De£nition at line 174 of £le ybos.h.

2.22.1.5 #de£ne D_RECORD 1

YBOS

De£nition at line 173 of £le ybos.h.

2.22.1.6 #de£ne DSP_ASC 3

Display data in ASCII format

De£nition at line 195 of £le ybos.h.

2.22.1.7 #de£ne DSP_BANK 2

Display data in bank format

De£nition at line 188 of £le ybos.h.

2.22.1.8 #de£ne DSP_DEC 1

Display data in decimal format

De£nition at line 193 of £le ybos.h.

2.22.1.9 #de£ne DSP_HEX 2

Display data in headecimal format

De£nition at line 194 of £le ybos.h.

2.22.1.10 #de£ne DSP_RAW 1

Display raw data

De£nition at line 187 of £le ybos.h.

2.22.1.11 #de£ne DSP_UNK 0

Display format unknown

De£nition at line 192 of £le ybos.h.

2.22.1.12 #de£ne F4_BKTYPE 4

Float 4 bytes

De£nition at line 344 of £le ybos.h.

2.22.1.13 #de£ne H_BLOCK_NUM 1

YBOS

De£nition at line 167 of £le ybos.h.

2.22.1.14 #de£ne H_BLOCK_SIZE 0

YBOS

De£nition at line 166 of £le ybos.h.

2.22.1.15 #de£ne H_HEAD_LEN 2

YBOS

De£nition at line 168 of £le ybos.h.

2.22.1.16 #de£ne H_START 3

YBOS

De£nition at line 169 of £le ybos.h.

2.22.1.17 #de£ne I1_BKTYPE 8

Signed Integer 1 byte

De£nition at line 346 of £le ybos.h.

Referenced by update_odb().

2.22.1.18 #de£ne I2_BKTYPE 1

Signed Integer 2 bytes

De£nition at line 341 of £le ybos.h.

2.22.1.19 #de£ne I4_BKTYPE 3

Signed Interger 4bytes

De£nition at line 343 of £le ybos.h.

2.22.1.20 #de£ne MAX_BKTYPE I1_BKTYPE+1

delimiter

De£nition at line 347 of £le ybos.h.

2.22.1.21 #de£ne YB_ADD_RUN 1

YBOS

De£nition at line 183 of £le ybos.h.

2.22.1.22 #de£ne YB_BANKLIST_MAX 32

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

De£nition at line 135 of £le ybos.h.

Referenced by ybk_list().

2.22.1.23 #de£ne YB_COMPLETE 1

YBOS

De£nition at line 179 of £le ybos.h.

2.22.1.24 #de£ne YB_INCOMPLETE 2

YBOS

De£nition at line 180 of £le ybos.h.

2.22.1.25 #de£ne YB_NO_RECOVER -1

YBOS

De£nition at line 181 of £le ybos.h.

2.22.1.26 #de£ne YB_NO_RUN 0

YBOS

De£nition at line 182 of £le ybos.h.

2.22.1.27 #de£ne YB_STRING_BANKLIST_MAX YB_BANKLIST_MAX ∗ 4

to be used for xbk_list()

De£nition at line 137 of £le ybos.h.

2.22.1.28 #de£ne YBOS_BUFFER_SIZE 3∗(YBOS_PHYREC_SIZE<<2) + MAX_EVENT_SIZE + 128

in BYTES

De£nition at line 133 of £le ybos.h.

2.22.1.29 #de£ne YBOS_HEADER_LENGTH 4

De£nition at line 132 of £le ybos.h.

2.22.1.30 #de£ne YBOS_PHYREC_SIZE 8192

I∗4

De£nition at line 130 of £le ybos.h.

2.23 YBOS Macros

De£nes

- #de£ne SWAP_D2WORD(_d2w)
- #de£ne EVID_TRINAT

- #de£ne YBOS_EVID_BANK(__a, __b, __c,__d,__e)
- #de£ne MIDAS_EVID_BANK(__a, __b, __c,__d,__e)

2.23.1 De£ne Documentation

2.23.1.1 #de£ne 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 £lled with the Midas info and other user information.

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

Omitting the declaration of the pre-processor ¤ag 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 £rst 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] Speci£c 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] Speci£c 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);
```

De£nition at line 275 of £le ybos.h.

2.23.1.2 #de£ne 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#

De£nition at line 319 of £le ybos.h.

2.23.1.3 #de£ne SWAP_D2WORD(_d2w)

Value:

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

word swap (I4=I21I22 -> I4=I22I21

De£nition at line 208 of £le ybos.h.

2.23.1.4 #de£ne 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#

De£nition at line 305 of £le ybos.h.

2.23.1.5 #de£ne YBOS_EVID_EVENT_ID(e) ∗((WORD ∗)(e)+1)

De£nition at line 279 of £le ybos.h.

2.23.1.6 #de£ne YBOS_EVID_EVENT_NB(e) ∗((DWORD ∗)(e)+1)

De£nition at line 284 of £le ybos.h.

2.23.1.7 #de£ne YBOS_EVID_RUN_NUMBER(e) ∗((DWORD ∗)(e)+3)

De£nition at line 283 of £le ybos.h.

2.23.1.8 #de£ne YBOS_EVID_SERIAL(e) ∗((DWORD ∗)(e)+1)

De£nition at line 281 of £le ybos.h.

2.23.1.9 #de£ne YBOS_EVID_TIME(e) ∗((DWORD ∗)(e)+2)

De£nition at line 282 of £le ybos.h.

2.23.1.10 #de£ne YBOS_EVID_TRIGGER_MASK(e) ∗((WORD ∗)(e)+0)

De£nition at line 280 of £le ybos.h.

2.24 YBOS error code

De£nes

- #de£ne YB_SUCCESS 1
- #de£ne YB_EVENT_NOT_SWAPPED 2
- #de£ne YB_DONE 2

- #de£ne YB_WRONG_BANK_TYPE -100
- #de£ne YB_BANK_NOT_FOUND -101
- #de£ne YB_SWAP_ERROR -102
- #de£ne YB_NOMORE_SLOT -103
- #de£ne YB_UNKNOWN_FORMAT -104

2.24.1 De£ne Documentation

2.24.1.1 #de£ne YB_BANK_NOT_FOUND -101

Bank not found

De£nition at line 151 of £le ybos.h.

2.24.1.2 #de£ne YB_DONE 2

Operation complete

De£nition at line 149 of £le ybos.h.

2.24.1.3 #de£ne YB_EVENT_NOT_SWAPPED 2

Not swapped

De£nition at line 148 of £le ybos.h.

2.24.1.4 #de£ne YB_NOMORE_SLOT -103

No more space for fragment

De£nition at line 153 of £le ybos.h.

2.24.1.5 #de£ne YB_SUCCESS 1

Ok

De£nition at line 147 of £le ybos.h.

2.24.1.6 #de£ne YB_SWAP_ERROR -102

Error swapping

De£nition at line 152 of £le ybos.h.

2.24.1.7 #de£ne YB_UNKNOWN_FORMAT -104

Unknown format (see YBOS format)

De£nition at line 154 of £le ybos.h.

2.24.1.8 #de£ne YB_WRONG_BANK_TYPE -100

Wrong bank type (see YBOS Bank Types)

De£nition at line 150 of £le ybos.h.

2.25 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_£nd (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)

2.25.1 Function Documentation

2.25.1.1 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 £ll a bank. It is incremented with every value written to the bank and £nally points to a location just after the last byte of the bank. It is then passed to ybk_close() to £nish 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.

De£nition at line 545 of £le ybos.c.

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

De£ne 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 £rst. 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 £rst 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 £rst empty data location.

Returns:
void

De£nition at line 434 of £le ybos.c.

2.25.1.3 INT ybk_£nd (DWORD ∗ *plrl*, char ∗ *bkname*, DWORD ∗ *bklen*, DWORD ∗ *bktype*, void ∗∗ *pbk*)

Find the requested bank and return the in£rmation 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 £rst data of the found bank.

Returns:
YB_SUCCESS, YB_BANK_NOT_FOUND, YB_WRONG_BANK_TYPE

De£nition at line 639 of £le ybos.c.

2.25.1.4 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 £rst. See YBOS bank examples.

Parameters:

plrl pointer to the £rst DWORD of the event area of event

Returns:

void

De£nition at line 404 of £le ybos.c.

2.25.1.5 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 £rst data of the current bank.

Returns:

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

De£nition at line 725 of £le ybos.c.

Referenced by update_odb().

2.25.1.6 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 prede£ned 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.

De£nition at line 590 of £le ybos.c.

2.25.1.7 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 £rst data of the located bank.

Returns:

Number of DWORD in bank or YB_BANK_NOT_FOUND, YB_WRONG_-BANK_TYPE (<0)

De£nition at line 686 of £le ybos.c.

2.25.1.8 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

De£nition at line 575 of £le ybos.c.

2.26 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 ∗time)
- char ∗ cm_get_version ()
- 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 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 ∗perror, INT strsize, INT async_¤ag, INT debug_¤ag)
- 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)

2.26.1 Function Documentation

2.26.1.1 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

De£nition at line 1701 of £le midas.c.

Referenced by al_trigger_alarm(), cm_transition(), and db_save_xml().

2.26.1.2 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

De£nition at line 1969 of £le midas.c.

Referenced by cm_set_client_info().

2.26.1.3 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()

De£nition at line 3538 of £le midas.c.

Referenced by scheduler().

2.26.1.4 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" ¤ag 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 de£ned by each client.

Returns:

CM_SUCCESS

De£nition at line 5203 of £le midas.c.

Referenced by main().

2.26.1.5 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

De£nition at line 2743 of £le midas.c.

2.26.1.6 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 £rst 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 noti£cations 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 Speci£es 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 Speci£es the experiment to connect to. If this string is empty, the number of de£ned experiments in exptab is checked. If only one experiment is de£ned, the function automatically connects to this one. If more than one experiment is de£ned, 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

De£nition at line 2382 of £le midas.c.

Referenced by main().

2.26.1.7 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 speci£c host.

For internal use only.

De£nition at line 2404 of £le midas.c.

Referenced by cm_connect_experiment(), and main().

2.26.1.8 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

De£nition at line 1918 of £le midas.c.

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

2.26.1.9 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()

De£nition at line 2809 of £le midas.c.

2.26.1.10 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

De£nition at line 2823 of £le midas.c.

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

2.26.1.11 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

De£nition at line 4275 of £le midas.c.

2.26.1.12 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

De£nition at line 5125 of £le midas.c.

Referenced by main().

2.26.1.13 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

De£nition at line 2208 of £le midas.c.

Referenced by bm_open_buffer().

2.26.1.14 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 £rst 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 ¤ags.

```
#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

De£nition at line 2284 of £le midas.c.

Referenced by main().

2.26.1.15 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

De£nition at line 2968 of £le midas.c.

Referenced by al_trigger_alarm(), ana_end_of_run(), analyzer_init(), cm_connect_-client(), 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().

2.26.1.16 INT cm_get_path (char ∗ *path*)

Return the path name previously set with cm_set_path.

Parameters:

path Pathname

Returns:

CM_SUCCESS

De£nition at line 1790 of £le midas.c.

Referenced by cm_connect_experiment1(), cm_msg_log(), cm_msg_log1(), and cm_-msg_retrieve().

2.26.1.17 char∗ cm_get_version ()

Return version number of current MIDAS library as a string

Returns:
version number ∗ 100

De£nition at line 1759 of £le midas.c.

Referenced by cm_transition().

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

Return watchdog information about speci£c 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

De£nition at line 3189 of £le midas.c.

2.26.1.19 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

De£nition at line 3169 of £le midas.c.

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

2.26.1.20 INT cm_list_experiments (char ∗ *host_name*, char *exp_name*[MAX_-EXPERIMENT][NAME_LENGTH])

Connect to a MIDAS server and return all de£ned experiments in ∗exp_name[MAX_-EXPERIMENTS]

Parameters:

host_name Internet host name.

exp_name list of experiment names

Returns:

CM_SUCCESS, RPC_NET_ERROR

De£nition at line 2601 of £le midas.c.

Referenced by cm_select_experiment().

2.26.1.21 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

De£nition at line 3477 of £le midas.c.

2.26.1.22 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 noti£cation 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 £les before the other clients are contacted, and 800 for stop, so that the £les 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

De£nition at line 3348 of £le midas.c.

Referenced by main().

2.26.1.23 INT cm_scan_experiments (void)

Scan the "exptab" £le for MIDAS experiment names and save them for later use by rpc_server_accept(). The £le is £rst 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

De£nition at line 1826 of £le midas.c.

Referenced by cm_connect_experiment1(), and cm_list_experiments().

2.26.1.24 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

De£nition at line 2704 of £le midas.c.

Referenced by cm_connect_experiment1().

2.26.1.25 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

De£nition at line 2031 of £le midas.c.

Referenced by cm_connect_experiment1().

2.26.1.26 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

De£nition at line 2905 of £le midas.c.

Referenced by cm_connect_experiment1(), and cm_disconnect_experiment().

2.26.1.27 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

De£nition at line 1773 of £le midas.c.

Referenced by cm_connect_experiment1().

2.26.1.28 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

De£nition at line 3418 of £le midas.c.

2.26.1.29 INT cm_set_watchdog_params (BOOL *call_watchdog*, DWORD *time-out*)

Sets the internal watchdog ¤ags 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 speci£es 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 speci£ed 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

De£nition at line 3055 of £le midas.c.

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

2.26.1.30 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

De£nition at line 5035 of £le midas.c.

Referenced by cm_transition(), and main().

2.26.1.31 INT cm_synchronize (DWORD ∗ *seconds*)

Get time from MIDAS server and set local time.

Parameters:

seconds Time in seconds

Returns:

CM_SUCCESS

De£nition at line 1673 of £le midas.c.

Referenced by main().

2.26.1.32 INT cm_time (DWORD ∗ *time*)

Get time from ss_time on server.

Parameters:

time string

Returns:

CM_SUCCESS

De£nition at line 1719 of £le midas.c.

Referenced by cm_transition().

2.26.1.33 INT cm_transition (INT *transition*, INT *run_number*, char ∗ *perror*, INT *strsize*, INT *async_¤ag*, INT *debug_¤ag*)

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

Synchronous/Asynchronous ¤ag. 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 speci£ed 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_¤ag 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 ¤ush buffers and close disk £les 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.

perror returned error string.

strsize Size of error string.

async_¤ag SYNC: synchronization ¤ag (SYNC:wait completion, ASYNC: retun immediately)

debug_¤ag If 1 output debugging information, if 2 output via cm_msg().

Returns:

CM_SUCCESS, <error> error code from remote client

De£nition at line 3635 of £le midas.c.

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

2.26.1.34 INT cm_yield (INT *millisec*)

Central yield functions for clients. This routine should be called in an in£nite 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 speci£ed time-out, the routine returns. If millisec=-1, it only returns when receiving an RPC_SHUTDOWN message.

Returns:

CM_SUCCESS, RPC_SHUTDOWN

De£nition at line 4222 of £le midas.c.

Referenced by scan_fragment(), and scheduler().

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

De£nition at line 3590 of £le midas.c.

Referenced by cm_transition().

2.27 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_¤ag)
- INT bm_¤ush_cache (INT buffer_handle, INT async_¤ag)
- INT bm_receive_event (INT buffer_handle, void ∗destination, INT ∗buf_size, INT async_¤ag)
- INT bm_skip_event (INT buffer_handle)
- INT bm_push_event (char ∗buffer_name)
- INT bm_check_buffers ()
- INT bm_empty_buffers ()

2.27.1 Function Documentation

2.27.1.1 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

De£nition at line 7485 of £le midas.c.

Referenced by cm_yield().

2.27.1.2 INT bm_close_all_buffers (void)

Close all open buffers

Returns:
BM_SUCCESS

De£nition at line 4778 of £le midas.c.

Referenced by cm_disconnect_experiment(), and cm_set_client_info().

2.27.1.3 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

De£nition at line 4667 of £le midas.c.

Referenced by bm_close_all_buffers(), and source_unbooking().

2.27.1.4 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

De£nition at line 5742 of £le midas.c.

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

2.27.1.5 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 identi£er given by bm_request_event()

Returns:

BM_SUCCESS, BM_INVALID_HANDLE

De£nition at line 6049 of £le midas.c.

Referenced by bm_close_buffer(), and source_unbooking().

2.27.1.6 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 re¤ects the hardware change. Users can be fooled by looking at histograms which re¤ect 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 ¤ushing all buffers once every second.

Returns:
BM_SUCCESS

De£nition at line 7807 of £le midas.c.

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

2.27.1.7 INT bm_¤ush_cache (INT *buffer_handle*, INT *async_¤ag*)

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 ¤ow to the data of the next run.

Parameters:

buffer_handle Buffer handle obtained via bm_open_buffer()

async_¤ag Synchronous/asynchronous ¤ag. 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_¤ag == 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 or EVENT_BUFFER_SIZE in midas.h and recompile.

De£nition at line 6468 of £le midas.c.

Referenced by bm_send_event(), close_buffers(), scan_fragment(), scheduler(), send_-event(), and tr_stop().

2.27.1.8 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 4383 of ﬁle midas.c.

Referenced by bm_ﬂush_cache(), bm_push_event(), bm_receive_event(), and bm_-send_event().

2.27.1.9 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 EVENT_BUFFER_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, EVENT_BUFFER_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 Size of buffer in bytes

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 con¤icts with an existing buffer of different size
BM_INVALID_PARAM Invalid parameter

De£nition at line 4445 of £le midas.c.

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

2.27.1.10 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

De£nition at line 7213 of £le midas.c.

Referenced by bm_check_buffers().

2.27.1.11 INT bm_receive_event (INT *buffer_handle*, void ∗ *destination*, INT ∗ *buf_size*, INT *async_¤ag*)

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, EVENT_BUFFER_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_¤ag Synchronous/asynchronous ¤ag. 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

De£nition at line 6833 of £le midas.c.

Referenced by handFlush(), and source_scan().

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

Delete a previously placed request for a speci£c 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

De£nition at line 5976 of £le midas.c.

Referenced by bm_delete_request().

2.27.1.13 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 identi£ed 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 speci£es how many events to receive. A value of GET_ALL receives all events which match the speci£ed 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 speci£ed 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.

De£nition at line 5908 of £le midas.c.

Referenced by cm_msg_register(), and source_booking().

2.27.1.14 INT bm_send_event (INT *buffer_handle*, void ∗ *source*, INT *buf_size*, INT *async_¤ag*)

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 noti£ed 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, EVENT_BUFFER_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_¤ag Synchronous/asynchronous ¤ag. 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_¤ag == 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 or EVENT_BUFFER_SIZE in midas.h and recompile.

De£nition at line 6113 of £le midas.c.

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

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

Modi£es 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

De£nition at line 5645 of £le midas.c.

Referenced by register_equipment().

2.27.1.16 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

De£nition at line 7165 of £le midas.c.

2.28 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, char ∗£lename, INT line, const char ∗routine, const char ∗format,...)
- INT cm_msg1 (INT message_type, char ∗£lename, 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)

2.28.1 Function Documentation

2.28.1.1 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 de£ned in midas.h

string Error string

Returns:

CM_SUCCESS

De£nition at line 1042 of £le midas.c.

Referenced by cm_connect_experiment().

2.28.1.2 INT cm_msg (INT *message_type*, char ∗ *£lename*, 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).

£lename Name of source £le where error occured

line Line number where error occured

routine Routine name.

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

Returns:

CM_SUCCESS

De£nition at line 1288 of £le midas.c.

Referenced by al_trigger_alarm(), analyzer_init(), bk_list(), bm_close_buffer(), bm_¤ush_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(), close_buffers(), cm_check_client(), cm_check_deferred_-transition(), cm_cleanup(), cm_connect_experiment1(), 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_create_key(), db_create_link(), db_create_record(), db_delete_key1(), db_enum_-key(), db_£nd_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_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(), interrupt_routine(), load_fragment(), main(), register_equipment(), rpc_¤ush_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().

2.28.1.3 INT cm_msg1 (INT *message_type*, char ∗ *£lename*, 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 £le 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 £rst arg in the following example uses the prede£ned macro MINFO which handles automatically the £rst 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.

£lename Name of source £le where error occured

line Line number where error occured

facility Logging £le name

routine Routine name

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

Returns:

CM_SUCCESS

De£nition at line 1401 of £le midas.c.

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

Write message to logging £le. Called by cm_msg.

Attention:

May burn your £ngers

Parameters:

message_type Message type

message Message string

Returns:

CM_SUCCESS

De£nition at line 1104 of £le midas.c.

Referenced by cm_msg().

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

Write message to logging £le. Called by cm_msg().

For Internal use only.

Parameters:

message_type Message type

message Message string

facility Message facility, £lename in which messages will be written

Returns:

CM_SUCCESS

De£nition at line 1173 of £le midas.c.

Referenced by cm_msg1().

2.28.1.6 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

De£nition at line 1532 of £le midas.c.

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

Retrieve old messages from log £le

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 £ll

Returns:
CM_SUCCESS

De£nition at line 1562 of £le midas.c.

2.28.1.8 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 de£ned callback routine and second to the "SYSMSG" buffer.

A user de£ned 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

De£nition at line 1087 of £le midas.c.

Referenced by cm_connect_experiment1(), and main().

2.29 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_£nd (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)

2.29.1 Function Documentation

2.29.1.1 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 £ll a bank. It is incremented with every value written to the bank and £nally points to a location just after the last byte of the bank. It is then passed to bk_close() to £nish the bank creation

Parameters:

event pointer to current composed event

pdata pointer to the data

Returns:

number of bytes contained in bank

De£nition at line 12901 of £le midas.c.

Referenced by adc_calib(), adc_summing(), eb_user(), read_scaler_event(), read_-trigger_event(), and scaler_accum().

2.29.1.2 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 £ll a bank. It is incremented with every value written to the bank and £nally points to a location just after the last byte of the bank. It is then passed to the function bk_close() to £nish 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 de£ned in midas.h

pdata pointer to the data area of the newly created bank

Returns:

void

De£nition at line 12780 of £le midas.c.

Referenced by adc_calib(), adc_summing(), eb_user(), read_scaler_event(), read_-trigger_event(), and scaler_accum().

2.29.1.3 INT bk_£nd (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

De£nition at line 13048 of £le midas.c.

2.29.1.4 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 £rst.

Parameters:

event pointer to the area of event

De£nition at line 12698 of £le midas.c.

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

2.29.1.5 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 £rst.

Parameters:

event pointer to the area of event

Returns:

void

De£nition at line 12739 of £le midas.c.

2.29.1.6 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 £rst call to this function.

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

Returns:

Size of bank in bytes

De£nition at line 13130 of £le midas.c.

Referenced by bk_list(), and update_odb().

2.29.1.7 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.

De£nition at line 12955 of £le midas.c.

2.29.1.8 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

De£nition at line 12999 of £le midas.c.

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

2.29.1.9 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

De£nition at line 12753 of £le midas.c.

Referenced by read_scaler_event(), and read_trigger_event().

2.29.1.10 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 ¤ag which is set by bk_init() to identify the endian format of an event. If force is FALSE, this ¤ag 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 13207 of file midas.c.

Referenced by eb_mfragment_add(), and source_scan().

2.30 Midas Alarm Functions (al_xxx)

Functions

- INT al_trigger_alarm (char ∗alarm_name, char ∗alarm_message, char ∗default_class, char ∗cond_str, INT type)

2.30.1 Function Documentation

2.30.1.1 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

De£nition at line 16177 of £le midas.c.

2.31 Midas History Functions (hs_xxx)

Functions

- INT hs_set_path (char ∗path)
- INT hs_open_£le (DWORD ltime, char ∗suf£x, INT mode, int ∗fh)

2.31.1 Function Documentation

2.31.1.1 INT hs_open_£le (DWORD *ltime*, char ∗ *suf£x*, INT *mode*, int ∗ *fh*)

Open history £le belonging to certain date. Internal use only.

Parameters:
ltime Date for which a history £le should be opened.
suf£x File name suf£x like "hst", "idx", "idf"
mode R/W access mode
fh File handle

Returns:
HS_SUCCESS

De£nition at line 13346 of £le midas.c.

2.31.1.2 INT hs_set_path (char ∗ *path*)

Sets the path for future history £le accesses. Should be called before any other history function is called.

Parameters:
path Directory where history £les reside

Returns:
HS_SUCCESS

De£nition at line 13319 of £le midas.c.

2.32 Midas Elog Functions (el_xxx)

Functions

- INT el_submit (int run, char ∗author, char ∗type, char ∗system, char ∗subject, char ∗text, char ∗reply_to, char ∗encoding, char ∗a£lename1, char ∗buffer1, INT buffer_size1, char ∗a£lename2, char ∗buffer2, INT buffer_size2, char ∗a£lename3, char ∗buffer3, INT buffer_size3, char ∗tag, INT tag_size)

2.32.1 Function Documentation

2.32.1.1 INT el_submit (int *run*, char ∗ *author*, char ∗ *type*, char ∗ *system*, char ∗ *subject*, char ∗ *text*, char ∗ *reply_to*, char ∗ *encoding*, char ∗ *a£lename1*, char ∗ *buffer1*, INT *buffer_size1*, char ∗ *a£lename2*, char ∗ *buffer2*, INT *buffer_size2*, char ∗ *a£lename3*, char ∗ *buffer3*, INT *buffer_size3*, char ∗ *tag*, INT *tag_size*)

Submit an ELog entry.

Parameters:

run Run Number.

author Message author.

type Message type.

system Message system.

subject Subject.

text Message text.

reply_to In reply to this message.

encoding Text encoding, either HTML or plain.

a£lename1 File name of attachment.

buffer1 File contents.

buffer_size1 Size of buffer in bytes.

a£lename2 File name of attachment.

buffer2 File contents.

buffer_size2 Size of buffer in bytes.

a£lename3 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

De£nition at line 15137 of £le midas.c.

2.33 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_¤ag)
- INT rpc_¤ush_event ()

2.33.1 Function Documentation

2.33.1.1 INT rpc_¤ush_event ()

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

Returns:
RPC_SUCCESS, RPC_NET_ERROR

De£nition at line 10380 of £le midas.c.

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

2.33.1.2 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 de£nitions. The end of the list must be indicated by a function ID of zero.

name Name of this client

Returns:
RPC_SUCCESS

De£nition at line 8272 of £le midas.c.

2.33.1.3 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 de£nitions. 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

De£nition at line 8292 of £le midas.c.

Referenced by cm_connect_experiment1(), and rpc_register_client().

2.33.1.4 INT rpc_send_event (INT *buffer_handle*, void ∗ *source*, INT *buf_size*, INT *async_¤ag*)

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_¤ag*** SYNC / ASYNC ¤ag. 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).

Returns:
BM_INVALID_PARAM, BM_ASYNC_RETURN, RPC_SUCCESS, RPC_NET_ERROR, RPC_NO_CONNECTION, RPC_EXCEED_BUFFER

De£nition at line 10197 of £le midas.c.

Referenced by interrupt_routine(), scheduler(), and source_scan().

2.33.1.5 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

De£nition at line 9286 of £le midas.c.

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

2.34 Midas Dual Buffer Memory Functions (dm_xxx)

Functions

- INT dm_buffer_create (INT size, INT user_max_event_size)

2.34.1 Function Documentation

2.34.1.1 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

De£nition at line 17152 of £le midas.c.

Referenced by main().

2.35 System Functions (ss_xxx)

Functions

- 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.

De£nition at line 2255 of £le 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(), 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 in£nite (until another process calls ss_wake)

Returns:

SS_SUCCESS

De£nition at line 2481 of £le system.c.

Referenced by cm_shutdown(), main(), read_trigger_event(), and register_-equipment().

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

Execute command in a separate process, close all open £le 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);
}
...
\encode
@param command Command to execute.
@return SS_SUCCESS or ss_exec() return code
*/
INT ss_system(char *command)
{






   system(command);
   return SS_SUCCESS;


}

/**dox***************************************************************/
```

```
/* DOXYGEN_SHOULD_SKIP_THIS */

/********************************************************************/
/**
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
\code
...
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

De£nition at line 1695 of £le system.c.

Referenced by dm_buffer_create().

2.35.1.4 INT ss_thread_kill (midas_thread_t *thread_id*)

Destroys the thread identi£ed 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

De£nition at line 1769 of £le system.c.

2.35.1.5 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

De£nition at line 2322 of £le system.c.

Referenced by 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 De£ne
- group System Macros
- group System Structure Declaration

2.37 System De£ne

De£nes

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

2.37.1 De£ne Documentation

2.37.1.1 #de£ne DR_ASCII (1<<7)

•

De£nition at line 197 of £le msystem.h.

2.37.1.2 #de£ne DRF_G_FLOAT (1<<6)

•

De£nition at line 196 of £le msystem.h.

2.37.1.3 #de£ne DRF_IEEE (1<<5)

•

De£nition at line 195 of £le msystem.h.

2.37.1.4 #de£ne DRI_16 (1<<0)

•

De£nition at line 190 of £le msystem.h.

2.37.1.5 #de£ne DRI_32 (1<<1)

•

De£nition at line 191 of £le msystem.h.

2.37.1.6 #de£ne DRI_64 (1<<2)

•

De£nition at line 192 of £le msystem.h.

2.37.1.7 #de£ne DRI_BIG_ENDIAN (1<<4)

•

De£nition at line 194 of £le msystem.h.

2.37.1.8 #de£ne DRI_LITTLE_ENDIAN (1<<3)

•

De£nition at line 193 of £le msystem.h.

2.38 System Macros

De£nes

- #de£ne WORD_SWAP(x)
- #de£ne DWORD_SWAP(x)
- #de£ne QWORD_SWAP(x)

2.38.1 De£ne Documentation

2.38.1.1 #de£ne 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

De£nition at line 217 of £le msystem.h.

Referenced by bk_swap().

2.38.1.2 #de£ne 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

De£nition at line 227 of £le msystem.h.

Referenced by bk_swap().

2.38.1.3 #de£ne WORD_SWAP(x)

Value:

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

SWAP WORD macro

De£nition at line 210 of £le 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 De£ne
- group Midas RPC_LIST

2.41 RPC De£ne

De£nes

- #de£ne RPC_CM_SET_CLIENT_INFO 11000
- #de£ne RPC_CM_SET_WATCHDOG_PARAMS 11001
- #de£ne RPC_CM_CLEANUP 11002
- #de£ne RPC_CM_GET_WATCHDOG_INFO 11003
- #de£ne RPC_CM_MSG_LOG 11004

- #de£ne RPC_CM_EXECUTE 11005
- #de£ne RPC_CM_SYNCHRONIZE 11006
- #de£ne RPC_CM_ASCTIME 11007
- #de£ne RPC_CM_TIME 11008
- #de£ne RPC_CM_MSG 11009
- #de£ne RPC_CM_EXIST 11011
- #de£ne RPC_CM_MSG_RETRIEVE 11012
- #de£ne RPC_CM_MSG_LOG1 11013
- #de£ne RPC_BM_OPEN_BUFFER 11100
- #de£ne RPC_BM_CLOSE_BUFFER 11101
- #de£ne RPC_BM_CLOSE_ALL_BUFFERS 11102
- #de£ne RPC_BM_GET_BUFFER_INFO 11103
- #de£ne RPC_BM_GET_BUFFER_LEVEL 11104
- #de£ne RPC_BM_INIT_BUFFER_COUNTERS 11105
- #de£ne RPC_BM_SET_CACHE_SIZE 11106
- #de£ne RPC_BM_ADD_EVENT_REQUEST 11107
- #de£ne RPC_BM_REMOVE_EVENT_REQUEST 11108
- #de£ne RPC_BM_SEND_EVENT 11109
- #de£ne RPC_BM_FLUSH_CACHE 11110
- #de£ne RPC_BM_RECEIVE_EVENT 11111
- #de£ne RPC_BM_MARK_READ_WAITING 11112
- #de£ne RPC_BM_EMPTY_BUFFERS 11113
- #de£ne RPC_BM_SKIP_EVENT 11114
- #de£ne RPC_DB_OPEN_DATABASE 11200
- #de£ne RPC_DB_CLOSE_DATABASE 11201
- #de£ne RPC_DB_CLOSE_ALL_DATABASES 11202
- #de£ne RPC_DB_CREATE_KEY 11203
- #de£ne RPC_DB_CREATE_LINK 11204
- #de£ne RPC_DB_SET_VALUE 11205
- #de£ne RPC_DB_GET_VALUE 11206
- #de£ne RPC_DB_FIND_KEY 11207
- #de£ne RPC_DB_FIND_LINK 11208
- #de£ne RPC_DB_GET_PATH 11209
- #de£ne RPC_DB_DELETE_KEY 11210
- #de£ne RPC_DB_ENUM_KEY 11211
- #de£ne RPC_DB_GET_KEY 11212
- #de£ne RPC_DB_GET_DATA 11213
- #de£ne RPC_DB_SET_DATA 11214
- #de£ne RPC_DB_SET_DATA_INDEX 11215
- #de£ne RPC_DB_SET_MODE 11216
- #de£ne RPC_DB_GET_RECORD_SIZE 11219
- #de£ne RPC_DB_GET_RECORD 11220
- #de£ne RPC_DB_SET_RECORD 11221

- #de£ne RPC_DB_ADD_OPEN_RECORD 11222
- #de£ne RPC_DB_REMOVE_OPEN_RECORD 11223
- #de£ne RPC_DB_SAVE 11224
- #de£ne RPC_DB_LOAD 11225
- #de£ne RPC_DB_SET_CLIENT_NAME 11226
- #de£ne RPC_DB_RENAME_KEY 11227
- #de£ne RPC_DB_ENUM_LINK 11228
- #de£ne RPC_DB_REORDER_KEY 11229
- #de£ne RPC_DB_CREATE_RECORD 11230
- #de£ne RPC_DB_GET_DATA_INDEX 11231
- #de£ne RPC_DB_GET_KEY_TIME 11232
- #de£ne RPC_DB_GET_OPEN_RECORDS 11233
- #de£ne RPC_DB_FLUSH_DATABASE 11235
- #de£ne RPC_DB_SET_DATA_INDEX2 11236
- #de£ne RPC_DB_GET_KEY_INFO 11237
- #de£ne RPC_DB_GET_DATA1 11238
- #de£ne RPC_DB_SET_NUM_VALUES 11239
- #de£ne RPC_DB_CHECK_RECORD 11240
- #de£ne RPC_DB_GET_NEXT_LINK 11241
- #de£ne RPC_HS_SET_PATH 11300
- #de£ne RPC_HS_DEFINE_EVENT 11301
- #de£ne RPC_HS_WRITE_EVENT 11302
- #de£ne RPC_HS_COUNT_EVENTS 11303
- #de£ne RPC_HS_ENUM_EVENTS 11304
- #de£ne RPC_HS_COUNT_VARS 11305
- #de£ne RPC_HS_ENUM_VARS 11306
- #de£ne RPC_HS_READ 11307
- #de£ne RPC_HS_GET_VAR 11308
- #de£ne RPC_HS_GET_EVENT_ID 11309
- #de£ne RPC_EL_SUBMIT 11400
- #de£ne RPC_AL_CHECK 11500
- #de£ne RPC_AL_TRIGGER_ALARM 11501
- #de£ne RPC_RC_TRANSITION 12000
- #de£ne RPC_ANA_CLEAR_HISTOS 13000
- #de£ne RPC_LOG_REWIND 14000
- #de£ne RPC_TEST 15000
- #de£ne RPC_CNAF16 16000
- #de£ne RPC_CNAF24 16001
- #de£ne RPC_MANUAL_TRIG 17000
- #de£ne RPC_ID_WATCHDOG 99997
- #de£ne RPC_ID_SHUTDOWN 99998
- #de£ne RPC_ID_EXIT 99999

2.41.1 De£ne Documentation

2.41.1.1 #de£ne RPC_AL_CHECK 11500

-

De£nition at line 181 of £le mrpc.h.

2.41.1.2 #de£ne RPC_AL_TRIGGER_ALARM 11501

-

De£nition at line 182 of £le mrpc.h.

Referenced by al_trigger_alarm().

2.41.1.3 #de£ne RPC_ANA_CLEAR_HISTOS 13000

-

De£nition at line 186 of £le mrpc.h.

2.41.1.4 #de£ne RPC_BM_ADD_EVENT_REQUEST 11107

-

De£nition at line 119 of £le mrpc.h.

2.41.1.5 #de£ne RPC_BM_CLOSE_ALL_BUFFERS 11102

-

De£nition at line 114 of £le mrpc.h.

Referenced by bm_close_all_buffers().

2.41.1.6 #de£ne RPC_BM_CLOSE_BUFFER 11101

•

De£nition at line 113 of £le mrpc.h.

Referenced by bm_close_buffer().

2.41.1.7 #de£ne RPC_BM_EMPTY_BUFFERS 11113

•

De£nition at line 125 of £le mrpc.h.

Referenced by bm_empty_buffers().

2.41.1.8 #de£ne RPC_BM_FLUSH_CACHE 11110

•

De£nition at line 122 of £le mrpc.h.

Referenced by bm_¤ush_cache().

2.41.1.9 #de£ne RPC_BM_GET_BUFFER_INFO 11103

•

De£nition at line 115 of £le mrpc.h.

2.41.1.10 #de£ne RPC_BM_GET_BUFFER_LEVEL 11104

•

De£nition at line 116 of £le mrpc.h.

2.41.1.11 #de£ne RPC_BM_INIT_BUFFER_COUNTERS 11105

•

De£nition at line 117 of £le mrpc.h.

2.41.1.12 #de£ne RPC_BM_MARK_READ_WAITING 11112

•

De£nition at line 124 of £le mrpc.h.

2.41.1.13 #de£ne RPC_BM_OPEN_BUFFER 11100

•

De£nition at line 112 of £le mrpc.h.

Referenced by bm_open_buffer().

2.41.1.14 #de£ne RPC_BM_RECEIVE_EVENT 11111

•

De£nition at line 123 of £le mrpc.h.

Referenced by bm_receive_event().

2.41.1.15 #de£ne RPC_BM_REMOVE_EVENT_REQUEST 11108

•

De£nition at line 120 of £le mrpc.h.

Referenced by bm_remove_event_request().

2.41.1.16 #de£ne RPC_BM_SEND_EVENT 11109

•

De£nition at line 121 of £le mrpc.h.

Referenced by bm_send_event(), and rpc_send_event().

2.41.1.17 #de£ne RPC_BM_SET_CACHE_SIZE 11106

•

De£nition at line 118 of £le mrpc.h.

Referenced by bm_set_cache_size().

2.41.1.18 #de£ne RPC_BM_SKIP_EVENT 11114

•

De£nition at line 126 of £le mrpc.h.

Referenced by bm_skip_event().

2.41.1.19 #de£ne RPC_CM_ASCTIME 11007

•

De£nition at line 105 of £le mrpc.h.

Referenced by cm_asctime().

2.41.1.20 #de£ne RPC_CM_CLEANUP 11002

•

De£nition at line 100 of £le mrpc.h.

Referenced by cm_cleanup().

2.41.1.21 #de£ne RPC_CM_EXECUTE 11005

•

De£nition at line 103 of £le mrpc.h.

Referenced by cm_execute().

2.41.1.22 #de£ne RPC_CM_EXIST 11011

•

De£nition at line 108 of £le mrpc.h.

Referenced by cm_exist().

2.41.1.23 #de£ne RPC_CM_GET_WATCHDOG_INFO 11003

•

De£nition at line 101 of £le mrpc.h.

Referenced by cm_get_watchdog_info().

2.41.1.24 #de£ne RPC_CM_MSG 11009

•

De£nition at line 107 of £le mrpc.h.

2.41.1.25 #de£ne RPC_CM_MSG_LOG 11004

•

De£nition at line 102 of £le mrpc.h.

Referenced by cm_msg_log().

2.41.1.26 #de£ne RPC_CM_MSG_LOG1 11013

•

De£nition at line 110 of £le mrpc.h.

Referenced by cm_msg_log1().

2.41.1.27 #de£ne RPC_CM_MSG_RETRIEVE 11012

•

De£nition at line 109 of £le mrpc.h.

Referenced by cm_msg_retrieve().

2.41.1.28 #de£ne RPC_CM_SET_CLIENT_INFO 11000

•

De£nition at line 98 of £le mrpc.h.

Referenced by cm_set_client_info().

2.41.1.29 #de£ne RPC_CM_SET_WATCHDOG_PARAMS 11001

•

De£nition at line 99 of £le mrpc.h.

Referenced by cm_set_watchdog_params().

2.41.1.30 #de£ne RPC_CM_SYNCHRONIZE 11006

-

De£nition at line 104 of £le mrpc.h.

Referenced by cm_synchronize().

2.41.1.31 #de£ne RPC_CM_TIME 11008

-

De£nition at line 106 of £le mrpc.h.

Referenced by cm_time().

2.41.1.32 #de£ne RPC_CNAF16 16000

-

De£nition at line 192 of £le mrpc.h.

Referenced by main().

2.41.1.33 #de£ne RPC_CNAF24 16001

-

De£nition at line 193 of £le mrpc.h.

Referenced by main().

2.41.1.34 #de£ne RPC_DB_ADD_OPEN_RECORD 11222

-

De£nition at line 148 of £le mrpc.h.

2.41.1.35 #de£ne RPC_DB_CHECK_RECORD 11240

-

De£nition at line 165 of £le mrpc.h.

Referenced by db_check_record().

2.41.1.36 #de£ne RPC_DB_CLOSE_ALL_DATABASES 11202

•

De£nition at line 130 of £le mrpc.h.

2.41.1.37 #de£ne RPC_DB_CLOSE_DATABASE 11201

•

De£nition at line 129 of £le mrpc.h.

Referenced by db_close_database().

2.41.1.38 #de£ne RPC_DB_CREATE_KEY 11203

•

De£nition at line 131 of £le mrpc.h.

Referenced by db_create_key().

2.41.1.39 #de£ne RPC_DB_CREATE_LINK 11204

•

De£nition at line 132 of £le mrpc.h.

Referenced by db_create_link().

2.41.1.40 #de£ne RPC_DB_CREATE_RECORD 11230

•

De£nition at line 156 of £le mrpc.h.

Referenced by db_create_record().

2.41.1.41 #de£ne RPC_DB_DELETE_KEY 11210

•

De£nition at line 138 of £le mrpc.h.

Referenced by db_delete_key().

2.41.1.42 #de£ne RPC_DB_ENUM_KEY 11211

•

De£nition at line 139 of £le mrpc.h.

Referenced by db_enum_key().

2.41.1.43 #de£ne RPC_DB_ENUM_LINK 11228

•

De£nition at line 154 of £le mrpc.h.

2.41.1.44 #de£ne RPC_DB_FIND_KEY 11207

•

De£nition at line 135 of £le mrpc.h.

Referenced by db_£nd_key().

2.41.1.45 #de£ne RPC_DB_FIND_LINK 11208

•

De£nition at line 136 of £le mrpc.h.

2.41.1.46 #de£ne RPC_DB_FLUSH_DATABASE 11235

•

De£nition at line 160 of £le mrpc.h.

2.41.1.47 #de£ne RPC_DB_GET_DATA 11213

•

De£nition at line 141 of £le mrpc.h.

Referenced by db_get_data().

2.41.1.48 #de£ne RPC_DB_GET_DATA1 11238

•

De£nition at line 163 of £le mrpc.h.

2.41.1.49 #de£ne RPC_DB_GET_DATA_INDEX 11231

•

De£nition at line 157 of £le mrpc.h.

Referenced by db_get_data_index().

2.41.1.50 #de£ne RPC_DB_GET_KEY 11212

•

De£nition at line 140 of £le mrpc.h.

Referenced by db_get_key().

2.41.1.51 #de£ne RPC_DB_GET_KEY_INFO 11237

•

De£nition at line 162 of £le mrpc.h.

Referenced by db_get_key_info().

2.41.1.52 #de£ne RPC_DB_GET_KEY_TIME 11232

•

De£nition at line 158 of £le mrpc.h.

Referenced by db_get_key_time().

2.41.1.53 #de£ne RPC_DB_GET_NEXT_LINK 11241

•

De£nition at line 166 of £le mrpc.h.

2.41.1.54 #de£ne RPC_DB_GET_OPEN_RECORDS 11233

-

De£nition at line 159 of £le mrpc.h.

2.41.1.55 #de£ne RPC_DB_GET_PATH 11209

-

De£nition at line 137 of £le mrpc.h.

2.41.1.56 #de£ne RPC_DB_GET_RECORD 11220

-

De£nition at line 146 of £le mrpc.h.

Referenced by db_get_record().

2.41.1.57 #de£ne RPC_DB_GET_RECORD_SIZE 11219

-

De£nition at line 145 of £le mrpc.h.

Referenced by db_get_record_size().

2.41.1.58 #de£ne RPC_DB_GET_VALUE 11206

-

De£nition at line 134 of £le mrpc.h.

Referenced by db_get_value().

2.41.1.59 #de£ne RPC_DB_LOAD 11225

-

De£nition at line 151 of £le mrpc.h.

Referenced by db_load().

2.41.1.60 #de£ne RPC_DB_OPEN_DATABASE 11200

-

De£nition at line 128 of £le mrpc.h.

Referenced by db_open_database().

2.41.1.61 #de£ne RPC_DB_REMOVE_OPEN_RECORD 11223

-

De£nition at line 149 of £le mrpc.h.

2.41.1.62 #de£ne RPC_DB_RENAME_KEY 11227

-

De£nition at line 153 of £le mrpc.h.

2.41.1.63 #de£ne RPC_DB_REORDER_KEY 11229

-

De£nition at line 155 of £le mrpc.h.

2.41.1.64 #de£ne RPC_DB_SAVE 11224

-

De£nition at line 150 of £le mrpc.h.

Referenced by db_save().

2.41.1.65 #de£ne RPC_DB_SET_CLIENT_NAME 11226

-

De£nition at line 152 of £le mrpc.h.

2.41.1.66 #de£ne RPC_DB_SET_DATA 11214

•

De£nition at line 142 of £le mrpc.h.

Referenced by db_set_data().

2.41.1.67 #de£ne RPC_DB_SET_DATA_INDEX 11215

•

De£nition at line 143 of £le mrpc.h.

Referenced by db_set_data_index().

2.41.1.68 #de£ne RPC_DB_SET_DATA_INDEX2 11236

•

De£nition at line 161 of £le mrpc.h.

2.41.1.69 #de£ne RPC_DB_SET_MODE 11216

•

De£nition at line 144 of £le mrpc.h.

2.41.1.70 #de£ne RPC_DB_SET_NUM_VALUES 11239

•

De£nition at line 164 of £le mrpc.h.

2.41.1.71 #de£ne RPC_DB_SET_RECORD 11221

•

De£nition at line 147 of £le mrpc.h.

Referenced by db_set_record().

2.41.1.72 #de£ne RPC_DB_SET_VALUE 11205

•

De£nition at line 133 of £le mrpc.h.

Referenced by db_set_value().

2.41.1.73 #de£ne RPC_EL_SUBMIT 11400

•

De£nition at line 179 of £le mrpc.h.

Referenced by el_submit().

2.41.1.74 #de£ne RPC_HS_COUNT_EVENTS 11303

•

De£nition at line 171 of £le mrpc.h.

2.41.1.75 #de£ne RPC_HS_COUNT_VARS 11305

•

De£nition at line 173 of £le mrpc.h.

2.41.1.76 #de£ne RPC_HS_DEFINE_EVENT 11301

•

De£nition at line 169 of £le mrpc.h.

2.41.1.77 #de£ne RPC_HS_ENUM_EVENTS 11304

•

De£nition at line 172 of £le mrpc.h.

2.41.1.78 #de£ne RPC_HS_ENUM_VARS 11306

-

De£nition at line 174 of £le mrpc.h.

2.41.1.79 #de£ne RPC_HS_GET_EVENT_ID 11309

-

De£nition at line 177 of £le mrpc.h.

2.41.1.80 #de£ne RPC_HS_GET_VAR 11308

-

De£nition at line 176 of £le mrpc.h.

2.41.1.81 #de£ne RPC_HS_READ 11307

-

De£nition at line 175 of £le mrpc.h.

2.41.1.82 #de£ne RPC_HS_SET_PATH 11300

-

De£nition at line 168 of £le mrpc.h.

Referenced by hs_set_path().

2.41.1.83 #de£ne RPC_HS_WRITE_EVENT 11302

-

De£nition at line 170 of £le mrpc.h.

2.41.1.84 #de£ne RPC_ID_EXIT 99999

•

De£nition at line 199 of £le mrpc.h.

2.41.1.85 #de£ne RPC_ID_SHUTDOWN 99998

•

De£nition at line 198 of £le mrpc.h.

2.41.1.86 #de£ne RPC_ID_WATCHDOG 99997

•

De£nition at line 197 of £le mrpc.h.

2.41.1.87 #de£ne RPC_LOG_REWIND 14000

•

De£nition at line 188 of £le mrpc.h.

2.41.1.88 #de£ne RPC_MANUAL_TRIG 17000

•

De£nition at line 195 of £le mrpc.h.

Referenced by register_equipment().

2.41.1.89 #de£ne RPC_RC_TRANSITION 12000

•

De£nition at line 184 of £le mrpc.h.

Referenced by cm_register_transition(), and cm_transition().

2.41.1.90 #de£ne RPC_TEST 15000

•

De£nition at line 190 of £le 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 ¤*ag*)

De£nition at line 1285 of £le 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

De£nition at line 142 of £le 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

De£nition at line 1269 of £le 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_£nd_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_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 index, 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 index, 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 index, 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_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 index, 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 socket)
- 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

De£nition at line 7431 of £le odb.c.

Referenced by 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

De£nition at line 7907 of £le 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

De£nition at line 1300 of £le 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

De£nition at line 7870 of £le 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

De£nition at line 5213 of £le odb.c.

Referenced by db_create_record(), and db_save().

2.44.1.6 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

De£nition at line 1716 of £le odb.c.

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

2.44.1.7 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

De£nition at line 1955 of £le odb.c.

2.44.1.8 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 modi£ed. 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

De£nition at line 7261 of £le odb.c.

Referenced by al_trigger_alarm(), analyzer_init(), cm_set_client_info(), db_check_-record(), and tr_start().

2.44.1.9 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

De£nition at line 2155 of £le odb.c.

Referenced by cm_set_client_info(), and db_create_record().

2.44.1.10 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

De£nition at line 1985 of £le odb.c.

Referenced by cm_delete_client_info(), and db_delete_key().

2.44.1.11 INT db_enum_key (HNDLE *hDB*, HNDLE *hKey*, INT *index*, 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.

index 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

De£nition at line 3385 of £le odb.c.

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

2.44.1.12 INT db_£nd_key (HNDLE *hDB*, HNDLE *hKey*, char ∗ *key_name*, HNDLE ∗ *subhKey*)

Returns key handle for a key with a speci£c 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_£nd_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

De£nition at line 2188 of £le odb.c.

Referenced by al_trigger_alarm(), analyzer_init(), cm_connect_client(), 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_set_value(), load_fragment(), logger_root(), register_equipment(), tr_start(), and update_odb().

2.44.1.13 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

De£nition at line 4145 of £le 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().

2.44.1.14 INT db_get_data_index (HNDLE *hDB*, HNDLE *hKey*, void ∗ *data*, INT ∗ *buf_size*, INT *index*, 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.

index 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

De£nition at line 4371 of £le odb.c.

Referenced by cm_transition().

2.44.1.15 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

De£nition at line 3717 of £le odb.c.

Referenced by 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().

2.44.1.16 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

De£nition at line 3835 of £le odb.c.

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

Get time when key was last modi£ed

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

De£nition at line 3777 of £le odb.c.

2.44.1.18 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

De£nition at line 6765 of £le odb.c.

Referenced by al_trigger_alarm(), cm_transition(), db_open_record(), db_update_-record(), register_equipment(), and tr_start().

2.44.1.19 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

De£nition at line 6679 of £le odb.c.

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

2.44.1.20 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

De£nition at line 3256 of £le odb.c.

Referenced by al_trigger_alarm(), ana_end_of_run(), 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(), register_equipment(), scheduler(), and tr_start().

2.44.1.21 INT db_load (HNDLE *hDB*, HNDLE *hKeyRoot*, char ∗ *£lename*, BOOL *bRemote*)

Load a branch of a database from an .ODB £le.

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.

£lename Filename of .ODB £le.

bRemote If TRUE, the £le 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

De£nition at line 5133 of £le odb.c.

2.44.1.22 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

De£nition at line 1580 of £le 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_£nd_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().

2.44.1.23 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

De£nition at line 988 of £le odb.c.

Referenced by cm_connect_experiment1().

2.44.1.24 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 modi£ed 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 modi£ed by other clients. In write mode, the local structure is copied to the ODB sub-tree if it has been modi£ed 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

De£nition at line 7736 of £le odb.c.

Referenced by analyzer_init(), cm_register_deferred_transition(), and register_-equipment().

2.44.1.25 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

De£nition at line 5468 of £le odb.c.

Referenced by db_create_record(), and db_load().

2.44.1.26 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

De£nition at line 1659 of £le odb.c.

2.44.1.27 INT db_save (HNDLE *hDB*, HNDLE *hKey*, char ∗ *£lename*, BOOL *bRemote*)

Save a branch of a database to an .ODB £le

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.

£lename Filename of .ODB £le.

bRemote Flag for saving database on remote server.

Returns:

DB_SUCCESS, DB_FILE_ERROR

De£nition at line 5873 of £le odb.c.

2.44.1.28 INT db_save_struct (HNDLE *hDB*, HNDLE *hKey*, char ∗ *£le_name*, char ∗ *struct_name*, BOOL *append*)

Save a branch of a database to a C structure .H £le

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

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

£le_name Filename of .ODB £le.

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

append If TRUE, append to end of existing £le

Returns:

DB_SUCCESS, DB_INVALID_HANDLE, DB_FILE_ERROR

De£nition at line 6129 of £le odb.c.

2.44.1.29 INT db_save_xml (HNDLE *hDB*, HNDLE *hKey*, char ∗ *£lename*)

Save a branch of a database to an .xml £le

This function is used by the ODBEdit command save to write the contents of the ODB into a XML £le. 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.

£lename Filename of .XML £le.

Returns:

DB_SUCCESS, DB_FILE_ERROR

De£nition at line 6089 of £le odb.c.

2.44.1.30 INT db_save_xml_key (HNDLE *hDB*, HNDLE *hKey*, INT *level*, INT *fh*)

De£nition at line 5971 of £le odb.c.

Referenced by db_save_xml().

2.44.1.31 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 8150 of file odb.c.

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

2.44.1.32 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

De£nition at line 4498 of £le odb.c.

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

2.44.1.33 INT db_set_data_index (HNDLE *hDB*, HNDLE *hKey*, void ∗ *data*, INT *data_size*, INT *index*, 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

index 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

De£nition at line 4722 of £le odb.c.

Referenced by cm_register_transition().

2.44.1.34 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

De£nition at line 6869 of £le odb.c.

Referenced by al_trigger_alarm(), db_open_record(), db_send_changed_records(), register_equipment(), and update_odb().

2.44.1.35 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

De£nition at line 3123 of £le 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().

2.44.1.36 INT db_sprintf (char ∗ *string*, void ∗ *data*, INT *data_size*, INT *index*, 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.

index Index for array data.

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

Returns:

DB_SUCCESS

De£nition at line 6303 of £le odb.c.

Referenced by db_copy(), and db_save_xml_key().

2.44.1.37 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

De£nition at line 1630 of £le 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_£nd_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().

2.44.1.38 INT db_update_record (INT *hDB*, INT *hKey*, int *socket*)

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 noti£cation to the client.

Parameters:

hDB ODB handle obtained via cm_get_experiment_database().

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

socket optional server socket

Returns:

DB_SUCCESS, DB_INVALID_HANDLE

De£nition at line 7942 of £le odb.c.

2.44.1.39 BOOL equal_ustring (char ∗ *str1*, char ∗ *str2*)

De£nition at line 1688 of £le odb.c.

Referenced by 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_£nd_key(), db_open_-database(), db_paste(), logger_root(), and register_equipment().

2.44.1.40 char∗ extract_key (char ∗ *key_list*, char ∗ *key_name*)

De£nition at line 1676 of £le odb.c.

Referenced by db_create_key(), and db_£nd_key().

2.44.1.41 void xml_encode (char ∗ *src*, int *size*)

De£nition at line 5928 of £le odb.c.

Referenced by db_save_xml_key().

3 Midas Directory Documentation

3.1 /home1/midas/midas-1.9.5/drivers/bus/ Directory Reference

Files

- £le esone.c

3.2 /home1/midas/midas-1.9.5/drivers/ Directory Reference

Directories

- directorybus

3.3 /home1/midas/midas-1.9.5/examples/eventbuilder/ Directory Reference

Files

- £le ebuser.c
- £le mevb.c

3.4 /home1/midas/midas-1.9.5/examples/ Directory Reference

Directories

- directoryeventbuilder
- directoryexperiment

3.5 /home1/midas/midas-1.9.5/examples/experiment/ Directory Reference

Files

- £le adccalib.c
- £le adcsum.c
- £le analyzer.c
- £le experim.h
- £le frontend.c
- £le scaler.c

3.6 /home1/midas/midas-1.9.5/include/ Directory Reference

Files

- £le mcstd.h
- £le midas.h
- £le mrpc.h
- £le msystem.h

- £le mvmestd.h
- £le ybos.h

3.7 /home1/midas/midas-1.9.5/src/ Directory Reference

Files

- £le mfe.c
- £le midas.c
- £le mrpc.c
- £le odb.c
- £le system.c
- £le 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

De£nition at line 43 of £le experim.h.

Referenced by adc_calib().

4.1.1.2 INT ADC_CALIBRATION_PARAM::pedestal[8]

De£nition at line 41 of £le experim.h.

Referenced by adc_calib().

4.1.1.3 ¤oat ADC_CALIBRATION_PARAM::software_gain[8]

De£nition at line 42 of £le experim.h.

Referenced by adc_calib().

4.2 ADC_SUMMING_PARAM Struct Reference

4.2.1 Field Documentation

4.2.1.1 ¤oat ADC_SUMMING_PARAM::adc_threshold

De£nition at line 77 of £le experim.h.

Referenced by adc_summing().

4.3 ALARM Struct Reference

4.3.1 Detailed Description

Alarm structure

De£nition at line 1683 of £le midas.h.

4.3.2 Field Documentation

4.3.2.1 BOOL ALARM::active

De£nition at line 1684 of £le midas.h.

Referenced by al_trigger_alarm().

4.3.2.2 char ALARM::alarm_class[32]

De£nition at line 1692 of £le midas.h.

Referenced by al_trigger_alarm().

4.3.2.3 char ALARM::alarm_message[80]

De£nition at line 1693 of £le midas.h.

Referenced by al_trigger_alarm().

4.3.2.4 INT ALARM::check_interval

De£nition at line 1687 of £le midas.h.

Referenced by al_trigger_alarm().

4.3.2.5 DWORD ALARM::checked_last

De£nition at line 1688 of £le midas.h.

Referenced by al_trigger_alarm().

4.3.2.6 char ALARM::condition[256]

De£nition at line 1691 of £le midas.h.

4.3.2.7 char ALARM::time_triggered_£rst[32]

De£nition at line 1689 of £le midas.h.

Referenced by al_trigger_alarm().

4.3.2.8 char ALARM::time_triggered_last[32]

De£nition at line 1690 of £le midas.h.

Referenced by al_trigger_alarm().

4.3.2.9 INT ALARM::triggered

De£nition at line 1685 of £le midas.h.

Referenced by al_trigger_alarm().

4.3.2.10 INT ALARM::type

De£nition at line 1686 of £le midas.h.

Referenced by al_trigger_alarm().

4.4 ALARM_CLASS Struct Reference

4.4.1 Detailed Description

Alarm class structure

De£nition at line 1653 of £le midas.h.

4.4.2 Field Documentation

4.4.2.1 char ALARM_CLASS::display_bgcolor[32]

De£nition at line 1662 of £le midas.h.

4.4.2.2 char ALARM_CLASS::display_fgcolor[32]

De£nition at line 1663 of £le midas.h.

4.4.2.3 char ALARM_CLASS::execute_command[256]

De£nition at line 1658 of £le midas.h.

4.4.2.4 INT ALARM_CLASS::execute_interval

De£nition at line 1659 of £le midas.h.

4.4.2.5 DWORD ALARM_CLASS::execute_last

De£nition at line 1660 of £le midas.h.

4.4.2.6 BOOL ALARM_CLASS::stop_run

De£nition at line 1661 of £le midas.h.

4.4.2.7 INT ALARM_CLASS::system_message_interval

De£nition at line 1656 of £le midas.h.

4.4.2.8 DWORD ALARM_CLASS::system_message_last

De£nition at line 1657 of £le midas.h.

4.4.2.9 BOOL ALARM_CLASS::write_elog_message

De£nition at line 1655 of £le midas.h.

4.4.2.10 BOOL ALARM_CLASS::write_system_message

De£nition at line 1654 of £le 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

De£nition at line 1453 of £le 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 ¤ag

De£nition at line 1463 of £le 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

De£nition at line 1464 of £le 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

De£nition at line 1462 of £le midas.h.

4.5.1.10 char ANA_MODULE::name[NAME_LENGTH]

Module name

De£nition at line 1452 of £le midas.h.

4.5.1.11 INT ANA_MODULE::param_size

Size of parameter structure

De£nition at line 1461 of £le midas.h.

4.5.1.12 void∗ ANA_MODULE::parameters

Pointer to parameter structure

De£nition at line 1460 of £le midas.h.

4.6 ANA_TEST Struct Reference

4.6.1 Field Documentation

4.6.1.1 DWORD ANA_TEST::count

De£nition at line 1514 of £le midas.h.

4.6.1.2 char ANA_TEST::name[80]

De£nition at line 1512 of £le midas.h.

4.6.1.3 DWORD ANA_TEST::previous_count

De£nition at line 1515 of £le midas.h.

4.6.1.4 BOOL ANA_TEST::registered

De£nition at line 1513 of £le midas.h.

4.6.1.5 BOOL ANA_TEST::value

De£nition at line 1516 of £le midas.h.

4.7 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

4.7.1 Field Documentation

4.7.1.1 void∗ ANALYZE_REQUEST::addr

Buffer for CWNT £lling

De£nition at line 1497 of £le midas.h.

4.7.1.2 ANA_MODULE** ANALYZE_REQUEST::ana_module

List of analyzer modules

De£nition at line 1487 of £le midas.h.

4.7.1.3 INT(* ANALYZE_REQUEST::analyzer)(EVENT_HEADER *, void *)

Pointer to user analyzer routine

4.7.1.4 AR_INFO ANALYZE_REQUEST::ar_info

From above

De£nition at line 1485 of £le midas.h.

4.7.1.5 AR_STATS ANALYZE_REQUEST::ar_stats

De£nition at line 1505 of £le midas.h.

4.7.1.6 BANK_LIST* ANALYZE_REQUEST::bank_list

List of banks for event

De£nition at line 1488 of £le midas.h.

4.7.1.7 HNDLE ANALYZE_REQUEST::buffer_handle

MIDAS buffer handle

De£nition at line 1493 of £le midas.h.

4.7.1.8 char ANALYZE_REQUEST::event_name[NAME_LENGTH]

Event name

De£nition at line 1484 of £le midas.h.

4.7.1.9 DWORD ANALYZE_REQUEST::events_received

number of events sent

De£nition at line 1503 of £le midas.h.

4.7.1.10 DWORD ANALYZE_REQUEST::events_written

number of events written

De£nition at line 1504 of £le midas.h.

4.7.1.11 HNDLE ANALYZE_REQUEST::hkey_common

Key to common subtree

De£nition at line 1496 of £le midas.h.

4.7.1.12 HNDLE ANALYZE_REQUEST::hkey_variables

Key to variables subtree in ODB

De£nition at line 1495 of £le midas.h.

4.7.1.13 char** ANALYZE_REQUEST::init_string

De£nition at line 1491 of £le midas.h.

4.7.1.14 struct { ... } ANALYZE_REQUEST::number

Buffer for event number for CWNT

4.7.1.15 HNDLE ANALYZE_REQUEST::request_id

Event request handle

De£nition at line 1494 of £le midas.h.

4.7.1.16 DWORD ANALYZE_REQUEST::run

De£nition at line 1499 of £le midas.h.

4.7.1.17 INT ANALYZE_REQUEST::rwnt_buffer_size

Size in events of RW N-tuple buf

De£nition at line 1489 of £le midas.h.

4.7.1.18 DWORD ANALYZE_REQUEST::serial

De£nition at line 1500 of £le midas.h.

4.7.1.19 INT ANALYZE_REQUEST::status

One of FE_xxx

De£nition at line 1492 of £le midas.h.

4.7.1.20 DWORD ANALYZE_REQUEST::time

De£nition at line 1501 of £le midas.h.

4.7.1.21 BOOL ANALYZE_REQUEST::use_tests

Use tests for this event

De£nition at line 1490 of £le midas.h.

4.8 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]

4.8.1 Field Documentation

4.8.1.1 char AR_INFO::buffer[NAME_LENGTH]

Event buffer to send events into

De£nition at line 1471 of £le midas.h.

4.8.1.2 char AR_INFO::client_name[NAME_LENGTH]

Analyzer name

De£nition at line 1473 of £le midas.h.

4.8.1.3 BOOL AR_INFO::enabled

Enable ¤ag

De£nition at line 1472 of £le midas.h.

4.8.1.4 INT AR_INFO::event_id

Event ID associated with equipm.

De£nition at line 1468 of £le midas.h.

4.8.1.5 char AR_INFO::host[NAME_LENGTH]

Host on which analyzer is running

De£nition at line 1474 of £le midas.h.

4.8.1.6 INT AR_INFO::sampling_type

GET_ALL/GET_SOME

De£nition at line 1470 of £le midas.h.

4.8.1.7 INT AR_INFO::trigger_mask

Trigger mask

De£nition at line 1469 of £le midas.h.

4.9 AR_STATS Struct Reference

4.9.1 Field Documentation

4.9.1.1 double AR_STATS::events_per_sec

De£nition at line 1479 of £le midas.h.

4.9.1.2 double AR_STATS::events_received

De£nition at line 1478 of £le midas.h.

4.9.1.3 double AR_STATS::events_written

De£nition at line 1480 of £le midas.h.

4.10 ASUM_BANK Struct Reference

4.10.1 Field Documentation

4.10.1.1 ¤oat ASUM_BANK::average

De£nition at line 110 of £le experim.h.

Referenced by adc_summing().

4.10.1.2 ¤oat ASUM_BANK::sum

De£nition at line 109 of £le experim.h.

Referenced by adc_summing().

4.11 BANK Struct Reference

Data Fields

- char name [4]
- WORD type
- WORD data_size

4.11.1 Field Documentation

4.11.1.1 WORD BANK::data_size

-

De£nition at line 1418 of £le midas.h.

Referenced by bk_close(), bk_create(), bk_£nd(), bk_iterate(), bk_locate(), and bk_-swap().

4.11.1.2 char BANK::name[4]

-

De£nition at line 1416 of £le midas.h.

Referenced by bk_close(), bk_create(), bk_£nd(), bk_list(), bk_locate(), and update_-odb().

4.11.1.3 WORD BANK::type

•

De£nition at line 1417 of £le midas.h.

Referenced by bk_close(), bk_create(), bk_£nd(), bk_locate(), bk_swap(), and update_odb().

4.12 BANK32 Struct Reference

Data Fields

- char name [4]
- DWORD type
- DWORD data_size

4.12.1 Field Documentation

4.12.1.1 DWORD BANK32::data_size

•

De£nition at line 1424 of £le midas.h.

Referenced by bk_close(), bk_create(), bk_£nd(), bk_locate(), and bk_swap().

4.12.1.2 char BANK32::name[4]

•

De£nition at line 1422 of £le midas.h.

Referenced by bk_close(), bk_create(), bk_£nd(), bk_list(), bk_locate(), and update_-odb().

4.12.1.3 DWORD BANK32::type

•

De£nition at line 1423 of £le midas.h.

Referenced by bk_close(), bk_create(), bk_£nd(), bk_locate(), bk_swap(), and update_odb().

4.13 BANK_HEADER Struct Reference

Data Fields

- DWORD data_size
- DWORD ¤ags

4.13.1 Field Documentation

4.13.1.1 DWORD BANK_HEADER::data_size

Size in bytes

De£nition at line 1411 of £le midas.h.

Referenced by bk_£nd(), bk_swap(), eb_mfragment_add(), eb_yfragment_add(), and source_scan().

4.13.1.2 DWORD BANK_HEADER::¤ags

internal ¤ag

De£nition at line 1412 of £le midas.h.

Referenced by bk_swap().

4.14 BANK_LIST Struct Reference

Data Fields

- char name [9]
- WORD type
- DWORD size
- char ∗∗ init_str
- BOOL output_¤ag
- void ∗ addr
- DWORD n_data
- HNDLE def_key

4.14.1 Field Documentation

4.14.1.1 void* BANK_LIST::addr

•

De£nition at line 1439 of £le midas.h.

4.14.1.2 HNDLE BANK_LIST::def_key

•

De£nition at line 1441 of £le midas.h.

4.14.1.3 char** BANK_LIST::init_str

•

De£nition at line 1437 of £le midas.h.

Referenced by register_equipment().

4.14.1.4 DWORD BANK_LIST::n_data

•

De£nition at line 1440 of £le midas.h.

4.14.1.5 char BANK_LIST::name[9]

•

De£nition at line 1434 of £le midas.h.

Referenced by register_equipment().

4.14.1.6 BOOL BANK_LIST::output_¤ag

•

De£nition at line 1438 of £le midas.h.

4.14.1.7 DWORD BANK_LIST::size

•

De£nition at line 1436 of £le midas.h.

4.14.1.8 WORD BANK_LIST::type

•

De£nition at line 1435 of £le midas.h.

Referenced by register_equipment().

4.15 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

4.15.1 Field Documentation

4.15.1.1 BOOL BUFFER::attached

TRUE if buffer is attached

De£nition at line 1281 of £le midas.h.

Referenced by bm_check_buffers(), bm_close_buffer(), bm_empty_buffers(), bm_¤ush_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().

4.15.1.2 void* BUFFER::buffer_data

pointer to buffer data

De£nition at line 1284 of £le midas.h.

Referenced by bm_open_buffer().

4.15.1.3 BUFFER_HEADER* BUFFER::buffer_header

pointer to buffer header

De£nition at line 1283 of £le midas.h.

Referenced by bm_check_buffers(), bm_close_buffer(), bm_empty_buffers(), bm_¤ush_cache(), bm_open_buffer(), bm_push_event(), bm_receive_event(), bm_remove_event_request(), bm_send_event(), bm_skip_event(), cm_cleanup(), and cm_set_watchdog_params().

4.15.1.4 BOOL BUFFER::callback

callback de£ned for this buffer

De£nition at line 1296 of £le midas.h.

Referenced by bm_open_buffer(), and bm_push_event().

4.15.1.5 INT BUFFER::client_index

index to CLIENT str. in buf.

De£nition at line 1282 of £le midas.h.

Referenced by bm_close_buffer(), bm_empty_buffers(), bm_¤ush_cache(), bm_open_buffer(), bm_push_event(), bm_receive_event(), bm_remove_event_request(), bm_send_event(), bm_skip_event(), cm_cleanup(), and cm_set_watchdog_params().

4.15.1.6 INT BUFFER::index

connection index / tid

De£nition at line 1295 of £le midas.h.

Referenced by bm_check_buffers(), bm_close_buffer(), bm_empty_buffers(), bm_-open_buffer(), and cm_set_watchdog_params().

4.15.1.7 HNDLE BUFFER::mutex

mutex/semaphore handle

De£nition at line 1293 of £le midas.h.

4.15.1.8 char* BUFFER::read_cache

cache for burst read

De£nition at line 1285 of £le midas.h.

Referenced by bm_push_event(), bm_receive_event(), and bm_set_cache_size().

4.15.1.9 INT BUFFER::read_cache_rp

cache read pointer

De£nition at line 1287 of £le midas.h.

Referenced by bm_empty_buffers(), bm_push_event(), bm_receive_event(), bm_set_-cache_size(), and bm_skip_event().

4.15.1.10 INT BUFFER::read_cache_size

cache size in bytes

De£nition at line 1286 of £le midas.h.

Referenced by bm_close_buffer(), bm_push_event(), bm_receive_event(), and bm_-set_cache_size().

4.15.1.11 INT BUFFER::read_cache_wp

cache write pointer

De£nition at line 1288 of £le midas.h.

Referenced by bm_empty_buffers(), bm_push_event(), bm_receive_event(), bm_set_-cache_size(), and bm_skip_event().

4.15.1.12 INT BUFFER::shm_handle

handle to shared memory

De£nition at line 1294 of £le midas.h.

Referenced by bm_close_buffer(), and bm_open_buffer().

4.15.1.13 char∗ BUFFER::write_cache

cache for burst read

De£nition at line 1289 of £le midas.h.

Referenced by bm_¤ush_cache(), bm_send_event(), and bm_set_cache_size().

4.15.1.14 INT BUFFER::write_cache_rp

cache read pointer

De£nition at line 1291 of £le midas.h.

Referenced by bm_¤ush_cache(), and bm_set_cache_size().

4.15.1.15 INT BUFFER::write_cache_size

cache size in bytes

De£nition at line 1290 of £le midas.h.

Referenced by bm_close_buffer(), bm_¤ush_cache(), bm_send_event(), and bm_set_-cache_size().

4.15.1.16 INT BUFFER::write_cache_wp

cache write pointer

De£nition at line 1292 of £le midas.h.

Referenced by bm_¤ush_cache(), bm_send_event(), and bm_set_cache_size().

4.16 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
- ¤oat data_rate
- BOOL read_wait
- INT write_wait
- BOOL wake_up
- BOOL all_¤ag
- DWORD last_activity
- DWORD watchdog_timeout

4.16.1 Field Documentation

4.16.1.1 BOOL BUFFER_CLIENT::all_¤ag

at least one GET_ALL request

De£nition at line 1256 of £le midas.h.

Referenced by bm_remove_event_request().

4.16.1.2 ¤oat BUFFER_CLIENT::data_rate

data rate in kB/sec

De£nition at line 1252 of £le midas.h.

4.16.1.3 EVENT_REQUEST BUFFER_CLIENT::event_request[MAX_-EVENT_REQUESTS]

De£nition at line 1260 of £le midas.h.

Referenced by bm_¤ush_cache(), bm_push_event(), bm_receive_event(), bm_-remove_event_request(), and bm_send_event().

4.16.1.4 DWORD BUFFER_CLIENT::last_activity

time of last activity

De£nition at line 1257 of £le midas.h.

Referenced by bm_open_buffer(), cm_cleanup(), and cm_set_watchdog_params().

4.16.1.5 INT BUFFER_CLIENT::max_request_index

index of last request

De£nition at line 1248 of £le midas.h.

Referenced by bm_¤ush_cache(), bm_push_event(), bm_receive_event(), bm_-remove_event_request(), and bm_send_event().

4.16.1.6 char BUFFER_CLIENT::name[NAME_LENGTH]

name of client

De£nition at line 1242 of £le midas.h.

Referenced by bm_open_buffer(), and cm_cleanup().

4.16.1.7 INT BUFFER_CLIENT::num_received_events

no of received events

De£nition at line 1249 of £le midas.h.

4.16.1.8 INT BUFFER_CLIENT::num_sent_events

no of sent events

De£nition at line 1250 of £le midas.h.

4.16.1.9 INT BUFFER_CLIENT::num_waiting_events

no of waiting events

De£nition at line 1251 of £le midas.h.

Referenced by bm_send_event().

4.16.1.10 INT BUFFER_CLIENT::pid

process ID

De£nition at line 1243 of £le midas.h.

Referenced by bm_close_buffer(), bm_¤ush_cache(), bm_open_buffer(), bm_push_-event(), bm_receive_event(), bm_send_event(), and cm_cleanup().

4.16.1.11 INT BUFFER_CLIENT::port

UDP port for wake up

De£nition at line 1246 of £le midas.h.

Referenced by bm_close_buffer(), bm_¤ush_cache(), bm_open_buffer(), bm_push_-event(), bm_receive_event(), bm_send_event(), and cm_cleanup().

4.16.1.12 INT BUFFER_CLIENT::read_pointer

read pointer to buffer

De£nition at line 1247 of £le midas.h.

Referenced by bm_empty_buffers(), bm_¤ush_cache(), bm_open_buffer(), bm_push_-event(), bm_receive_event(), bm_send_event(), and bm_skip_event().

4.16.1.13 BOOL BUFFER_CLIENT::read_wait

wait for read - ¤ag

De£nition at line 1253 of £le midas.h.

Referenced by bm_close_buffer(), bm_¤ush_cache(), bm_receive_event(), bm_send_-event(), and cm_cleanup().

4.16.1.14 INT BUFFER_CLIENT::thandle

thread handle

De£nition at line 1245 of £le midas.h.

Referenced by bm_open_buffer().

4.16.1.15 INT BUFFER_CLIENT::tid

thread ID

De£nition at line 1244 of £le midas.h.

Referenced by bm_open_buffer(), bm_push_event(), and bm_receive_event().

4.16.1.16 BOOL BUFFER_CLIENT::wake_up

client got a wake-up msg

De£nition at line 1255 of £le midas.h.

4.16.1.17 DWORD BUFFER_CLIENT::watchdog_timeout

timeout in ms

De£nition at line 1258 of £le midas.h.

Referenced by bm_open_buffer(), cm_cleanup(), and cm_set_watchdog_params().

4.16.1.18 INT BUFFER_CLIENT::write_wait

wait for write # bytes

De£nition at line 1254 of £le midas.h.

Referenced by bm_close_buffer(), bm_¤ush_cache(), bm_push_event(), bm_receive_-event(), bm_send_event(), and cm_cleanup().

4.17 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]

4.17.1 Field Documentation

4.17.1.1 BUFFER_CLIENT BUFFER_HEADER::client[MAX_CLIENTS]

entries for clients

De£nition at line 1274 of £le midas.h.

Referenced by bm_close_buffer(), bm_¤ush_cache(), bm_open_buffer(), bm_push_-event(), bm_receive_event(), bm_send_event(), bm_skip_event(), cm_cleanup(), and cm_set_watchdog_params().

4.17.1.2 INT BUFFER_HEADER::max_client_index

index of last client

De£nition at line 1267 of £le midas.h.

Referenced by bm_close_buffer(), bm_¤ush_cache(), bm_open_buffer(), bm_push_-event(), bm_receive_event(), bm_send_event(), and cm_cleanup().

4.17.1.3 char BUFFER_HEADER::name[NAME_LENGTH]

name of buffer

De£nition at line 1265 of £le midas.h.

Referenced by bm_check_buffers(), bm_close_buffer(), bm_¤ush_cache(), bm_open_-buffer(), bm_push_event(), bm_send_event(), and cm_cleanup().

4.17.1.4 INT BUFFER_HEADER::num_clients

no of active clients

De£nition at line 1266 of £le midas.h.

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

4.17.1.5 INT BUFFER_HEADER::num_in_events

no of received events

De£nition at line 1271 of £le midas.h.

Referenced by bm_¤ush_cache(), and bm_send_event().

4.17.1.6 INT BUFFER_HEADER::num_out_events

no of distributed events

De£nition at line 1272 of £le midas.h.

Referenced by bm_push_event(), and bm_receive_event().

4.17.1.7 INT BUFFER_HEADER::read_pointer

read pointer

De£nition at line 1269 of £le midas.h.

Referenced by bm_¤ush_cache(), bm_push_event(), bm_receive_event(), and bm_-send_event().

4.17.1.8 INT BUFFER_HEADER::size

size of data area in bytes

De£nition at line 1268 of £le midas.h.

Referenced by bm_¤ush_cache(), bm_open_buffer(), bm_push_event(), bm_receive_-event(), and bm_send_event().

4.17.1.9 INT BUFFER_HEADER::write_pointer

read pointer

De£nition at line 1270 of £le midas.h.

Referenced by bm_¤ush_cache(), bm_open_buffer(), bm_push_event(), bm_receive_-event(), bm_send_event(), and bm_skip_event().

4.18 BUS_DRIVER Struct Reference

Data Fields

- char name [NAME_LENGTH]
- INT(∗ bd)(INT cmd,...)
- void ∗ bd_info

4.18.1 Field Documentation

4.18.1.1 INT(∗ BUS_DRIVER::bd)(INT cmd,...)

Device driver entry point

4.18.1.2 void∗ BUS_DRIVER::bd_info

Private info for bus driver

De£nition at line 1337 of £le midas.h.

4.18.1.3 char BUS_DRIVER::name[NAME_LENGTH]

Driver name

De£nition at line 1335 of £le midas.h.

4.19 DATABASE Struct Reference

4.19.1 Field Documentation

4.19.1.1 BOOL DATABASE::attached

De£nition at line 517 of £le 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_£nd_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().

4.19.1.2 INT DATABASE::client_index

De£nition at line 518 of £le msystem.h.

Referenced by cm_cleanup(), cm_set_watchdog_params(), db_close_database(), and db_open_database().

4.19.1.3 void* DATABASE::database_data

De£nition at line 520 of £le msystem.h.

Referenced by db_open_database().

4.19.1.4 DATABASE_HEADER* DATABASE::database_header

De£nition at line 519 of £le 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_£nd_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().

4.19.1.5 INT DATABASE::index

De£nition at line 524 of £le msystem.h.

Referenced by cm_set_watchdog_params(), db_close_database(), and db_open_-database().

4.19.1.6 INT DATABASE::lock_cnt

De£nition at line 522 of £le msystem.h.

Referenced by db_lock_database(), db_open_database(), and db_unlock_database().

4.19.1.7 HNDLE DATABASE::mutex

De£nition at line 521 of £le msystem.h.

4.19.1.8 char DATABASE::name[NAME_LENGTH]

De£nition at line 516 of £le msystem.h.

4.19.1.9 BOOL DATABASE::protect

De£nition at line 525 of £le msystem.h.

Referenced by db_lock_database(), db_open_database(), db_protect_database(), and db_unlock_database().

4.19.1.10 HNDLE DATABASE::shm_handle

De£nition at line 523 of £le msystem.h.

Referenced by db_close_database(), and db_open_database().

4.20 DATABASE_CLIENT Struct Reference

4.20.1 Field Documentation

4.20.1.1 DWORD DATABASE_CLIENT::last_activity

De£nition at line 490 of £le msystem.h.

Referenced by cm_cleanup(), cm_get_watchdog_info(), cm_set_watchdog_params(), and db_open_database().

4.20.1.2 INT DATABASE_CLIENT::max_index

De£nition at line 492 of £le msystem.h.

Referenced by db_close_database(), and db_open_database().

4.20.1.3 char DATABASE_CLIENT::name[NAME_LENGTH]

De£nition at line 484 of £le msystem.h.

Referenced by cm_cleanup(), cm_get_watchdog_info(), and db_open_database().

4.20.1.4 INT DATABASE_CLIENT::num_open_records

De£nition at line 489 of £le msystem.h.

Referenced by db_open_database().

4.20.1.5 OPEN_RECORD DATABASE_CLIENT::open_record[MAX_OPEN_-RECORDS]

De£nition at line 494 of £le msystem.h.

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

4.20.1.6 INT DATABASE_CLIENT::pid

De£nition at line 485 of £le msystem.h.

Referenced by cm_cleanup(), cm_get_watchdog_info(), db_close_database(), and db_open_database().

4.20.1.7 INT DATABASE_CLIENT::port

De£nition at line 488 of £le msystem.h.

Referenced by db_open_database().

4.20.1.8 INT DATABASE_CLIENT::thandle

De£nition at line 487 of £le msystem.h.

Referenced by db_open_database().

4.20.1.9 INT DATABASE_CLIENT::tid

De£nition at line 486 of £le msystem.h.

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

4.20.1.10 DWORD DATABASE_CLIENT::watchdog_timeout

De£nition at line 491 of £le msystem.h.

Referenced by cm_cleanup(), cm_get_watchdog_info(), cm_set_watchdog_params(), and db_open_database().

4.21 DATABASE_HEADER Struct Reference

4.21.1 Field Documentation

4.21.1.1 DATABASE_CLIENT DATABASE_HEADER::client[MAX_-CLIENTS]

De£nition at line 509 of £le msystem.h.

Referenced by cm_check_client(), cm_cleanup(), cm_get_watchdog_info(), cm_set_-watchdog_params(), db_close_database(), and db_open_database().

4.21.1.2 INT DATABASE_HEADER::data_size

De£nition at line 504 of £le msystem.h.

Referenced by db_close_database(), and db_open_database().

4.21.1.3 INT DATABASE_HEADER::£rst_free_data

De£nition at line 507 of £le msystem.h.

Referenced by db_open_database().

4.21.1.4 INT DATABASE_HEADER::£rst_free_key

De£nition at line 506 of £le msystem.h.

Referenced by db_open_database().

4.21.1.5 INT DATABASE_HEADER::key_size

De£nition at line 503 of £le msystem.h.

Referenced by db_open_database().

4.21.1.6 INT DATABASE_HEADER::max_client_index

De£nition at line 502 of £le msystem.h.

Referenced by cm_check_client(), cm_cleanup(), cm_get_watchdog_info(), db_-close_database(), and db_open_database().

4.21.1.7 char DATABASE_HEADER::name[NAME_LENGTH]

De£nition at line 499 of £le msystem.h.

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

4.21.1.8 INT DATABASE_HEADER::num_clients

De£nition at line 501 of £le msystem.h.

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

4.21.1.9 INT DATABASE_HEADER::root_key

Definition at line 505 of file msystem.h.

Referenced by db_create_key(), db_delete_key1(), db_enum_key(), db_find_key(), and db_open_database().

4.21.1.10 INT DATABASE_HEADER::version

Definition at line 500 of file msystem.h.

Referenced by db_open_database().

4.22 DEF_RECORD Struct Reference

4.22.1 Field Documentation

4.22.1.1 DWORD DEF_RECORD::def_offset

Definition at line 1550 of file midas.h.

4.22.1.2 DWORD DEF_RECORD::event_id

Definition at line 1548 of file midas.h.

4.22.1.3 char DEF_RECORD::event_name[NAME_LENGTH]

Definition at line 1549 of file midas.h.

4.23 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

4.23.1 Field Documentation

4.23.1.1 INT(∗ DEVICE_DRIVER::bd)(INT cmd,...)

Bus driver entry point

4.23.1.2 INT DEVICE_DRIVER::channels

Number of channels

De£nition at line 1343 of £le midas.h.

4.23.1.3 INT(∗ DEVICE_DRIVER::dd)(INT cmd,...)

Device driver entry point

4.23.1.4 void∗ DEVICE_DRIVER::dd_info

Private info for device driver

De£nition at line 1346 of £le midas.h.

4.23.1.5 DWORD DEVICE_DRIVER::¤ags

Combination of DF_xx

De£nition at line 1345 of £le midas.h.

4.23.1.6 char DEVICE_DRIVER::name[NAME_LENGTH]

Driver name

De£nition at line 1341 of £le midas.h.

Referenced by register_equipment().

4.24 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

4.24.1 Field Documentation

4.24.1.1 HNDLE eqpmnt::buffer_handle

MIDAS buffer handle

De£nition at line 1388 of £le midas.h.

Referenced by interrupt_routine(), register_equipment(), scheduler(), send_event(), and tr_stop().

4.24.1.2 DWORD eqpmnt::bytes_sent

number of bytes sent

De£nition at line 1394 of £le midas.h.

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

4.24.1.3 INT(∗ eqpmnt::cd)(INT cmd, PEQUIPMENT)

Class driver routine

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

4.24.1.4 void∗ eqpmnt::cd_info

private data for class driver

De£nition at line 1382 of £le midas.h.

4.24.1.5 DEVICE_DRIVER* eqpmnt::driver

Device driver list

De£nition at line 1380 of £le midas.h.

Referenced by register_equipment().

4.24.1.6 void* eqpmnt::event_descrip

Init string for £xed events or bank list

De£nition at line 1381 of £le midas.h.

Referenced by register_equipment().

4.24.1.7 DWORD eqpmnt::events_sent

number of events sent

De£nition at line 1395 of £le midas.h.

Referenced by display(), interrupt_routine(), scan_fragment(), scheduler(), send_-event(), source_scan(), and tr_stop().

4.24.1.8 INT eqpmnt::format

FORMAT_xxx

De£nition at line 1387 of £le midas.h.

Referenced by load_fragment(), register_equipment(), and scheduler().

4.24.1.9 HNDLE eqpmnt::hkey_variables

Key to variables subtree in ODB

De£nition at line 1389 of £le midas.h.

Referenced by register_equipment(), and scheduler().

4.24.1.10 EQUIPMENT_INFO eqpmnt::info

From above

De£nition at line 1377 of £le midas.h.

Referenced by display(), interrupt_routine(), load_fragment(), main(), register_-equipment(), scan_fragment(), scheduler(), send_all_periodic_events(), send_event(), and tr_start().

4.24.1.11 DWORD eqpmnt::last_called

Last time event was read

De£nition at line 1384 of £le midas.h.

Referenced by interrupt_routine(), scheduler(), and send_event().

4.24.1.12 DWORD eqpmnt::last_idle

Last time idle func was called

De£nition at line 1385 of £le midas.h.

Referenced by scheduler().

4.24.1.13 char eqpmnt::name[NAME_LENGTH]

Equipment name

De£nition at line 1376 of £le midas.h.

Referenced by display(), main(), register_equipment(), scheduler(), send_all_-periodic_events(), send_event(), tr_start(), and tr_stop().

4.24.1.14 DWORD eqpmnt::odb_in

updated ODB -> FE

De£nition at line 1393 of £le midas.h.

Referenced by tr_start().

4.24.1.15 DWORD eqpmnt::odb_out

updates FE -> ODB

De£nition at line 1392 of £le midas.h.

Referenced by interrupt_routine(), scheduler(), send_event(), and tr_start().

4.24.1.16 DWORD eqpmnt::poll_count

Needed to poll ’period’

De£nition at line 1386 of £le midas.h.

Referenced by register_equipment(), and scheduler().

4.24.1.17 INT(∗ eqpmnt::readout)(char ∗, INT)

Pointer to user readout routine

Referenced by interrupt_routine(), scheduler(), and send_event().

4.24.1.18 DWORD eqpmnt::serial_number

event serial number

De£nition at line 1390 of £le midas.h.

Referenced by eb_user(), interrupt_routine(), scheduler(), send_event(), source_scan(), and tr_start().

4.24.1.19 EQUIPMENT_STATS eqpmnt::stats

De£nition at line 1396 of £le midas.h.

Referenced by close_buffers(), display(), register_equipment(), scan_fragment(), scheduler(), send_event(), tr_start(), and tr_stop().

4.24.1.20 INT eqpmnt::status

One of FE_xxx

De£nition at line 1383 of £le midas.h.

Referenced by display(), main(), register_equipment(), scheduler(), and send_all_-periodic_events().

4.24.1.21 DWORD eqpmnt::subevent_number

subevent number

De£nition at line 1391 of £le midas.h.

Referenced by scheduler(), and tr_start().

4.25 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_£le_name [256]

4.25.1 Field Documentation

4.25.1.1 char EQUIPMENT_INFO::buffer[NAME_LENGTH]

Event buffer to send events into

De£nition at line 1352 of £le midas.h.

Referenced by register_equipment().

4.25.1.2 BOOL EQUIPMENT_INFO::enabled

Enable ¤ag

De£nition at line 1356 of £le midas.h.

Referenced by display(), register_equipment(), scheduler(), send_all_periodic_-events(), and tr_start().

4.25.1.3 INT EQUIPMENT_INFO::eq_type

One of EQ_xxx

De£nition at line 1353 of £le midas.h.

Referenced by main(), register_equipment(), scheduler(), and send_event().

4.25.1.4 WORD EQUIPMENT_INFO::event_id

Event ID associated with equipm.

De£nition at line 1350 of £le midas.h.

Referenced by interrupt_routine(), register_equipment(), scheduler(), send_event(), and source_scan().

4.25.1.5 double EQUIPMENT_INFO::event_limit

Stop run when limit is reached

De£nition at line 1359 of £le midas.h.

Referenced by register_equipment(), and scheduler().

4.25.1.6 char EQUIPMENT_INFO::format[8]

Data format to produce

De£nition at line 1355 of £le midas.h.

Referenced by load_fragment(), and register_equipment().

4.25.1.7 char EQUIPMENT_INFO::frontend_£le_name[256]

Source £le used for user FE

De£nition at line 1364 of £le midas.h.

Referenced by register_equipment().

4.25.1.8 char EQUIPMENT_INFO::frontend_host[NAME_LENGTH]

Host on which FE is running

De£nition at line 1362 of £le midas.h.

Referenced by register_equipment().

4.25.1.9 char EQUIPMENT_INFO::frontend_name[NAME_LENGTH]

Frontend name

De£nition at line 1363 of £le midas.h.

Referenced by register_equipment().

4.25.1.10 INT EQUIPMENT_INFO::history

Log history

De£nition at line 1361 of £le midas.h.

Referenced by interrupt_routine(), scheduler(), and send_event().

4.25.1.11 DWORD EQUIPMENT_INFO::num_subevents

Number of events in super event

De£nition at line 1360 of £le midas.h.

Referenced by scheduler().

4.25.1.12 INT EQUIPMENT_INFO::period

Readout interval/Polling time in ms

De£nition at line 1358 of £le midas.h.

Referenced by register_equipment(), and scheduler().

4.25.1.13 INT EQUIPMENT_INFO::read_on

Combination of Read-On ¤ags RO_xxx

De£nition at line 1357 of £le midas.h.

Referenced by interrupt_routine(), scheduler(), send_all_periodic_events(), and send_-event().

4.25.1.14 INT EQUIPMENT_INFO::source

Event source (LAM/IRQ)

De£nition at line 1354 of £le midas.h.

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

4.25.1.15 WORD EQUIPMENT_INFO::trigger_mask

Trigger mask

De£nition at line 1351 of £le midas.h.

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

4.26 EQUIPMENT_STATS Struct Reference

4.26.1 Field Documentation

4.26.1.1 double EQUIPMENT_STATS::events_per_sec

De£nition at line 1369 of £le midas.h.

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

4.26.1.2 double EQUIPMENT_STATS::events_sent

De£nition at line 1368 of £le midas.h.

Referenced by close_buffers(), display(), register_equipment(), scan_fragment(), scheduler(), send_event(), tr_start(), and tr_stop().

4.26.1.3 double EQUIPMENT_STATS::kbytes_per_sec

De£nition at line 1370 of £le midas.h.

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

4.27 EVENT_HEADER Struct Reference

4.27.1 Detailed Description

Event header

De£nition at line 1176 of £le midas.h.

Data Fields

- short int event_id
- short int trigger_mask
- DWORD serial_number
- DWORD time_stamp
- DWORD data_size

4.27.2 Field Documentation

4.27.2.1 DWORD EVENT_HEADER::data_size

size of event in bytes w/o header

De£nition at line 1181 of £le midas.h.

Referenced by bm_compose_event(), bm_¤ush_cache(), bm_push_event(), bm_-receive_event(), cm_msg(), cm_msg1(), eb_user(), interrupt_routine(), scheduler(), and send_event().

4.27.2.2 short int EVENT_HEADER::event_id

event ID starting from one

De£nition at line 1177 of £le midas.h.

Referenced by bm_compose_event(), bm_match_event(), bm_push_event(), bm_-receive_event(), interrupt_routine(), scheduler(), and send_event().

4.27.2.3 DWORD EVENT_HEADER::serial_number

serial number starting from one

De£nition at line 1179 of £le midas.h.

Referenced by bm_compose_event(), bm_receive_event(), eb_user(), interrupt_-routine(), scheduler(), and send_event().

4.27.2.4 DWORD EVENT_HEADER::time_stamp

time of production of event

De£nition at line 1180 of £le midas.h.

Referenced by bm_compose_event(), bm_receive_event(), interrupt_routine(), sched-uler(), and send_event().

4.27.2.5 short int EVENT_HEADER::trigger_mask

hardware trigger mask

De£nition at line 1178 of £le midas.h.

Referenced by bm_compose_event(), bm_match_event(), bm_receive_event(), interrupt_routine(), scheduler(), and send_event().

4.28 EVENT_REQUEST Struct Reference

4.28.1 Detailed Description

Buffer structure

De£nition at line 1231 of £le midas.h.

Data Fields

- INT id
- BOOL valid
- short int event_id

- short int trigger_mask
- INT sampling_type

4.28.2 Field Documentation

4.28.2.1 void(∗ EVENT_REQUEST::dispatch)(HNDLE, HNDLE, EVENT_HEADER ∗, void ∗)

4.28.2.2 short int EVENT_REQUEST::event_id

event ID

De£nition at line 1234 of £le midas.h.

Referenced by bm_¤ush_cache(), bm_push_event(), bm_receive_event(), and bm_-send_event().

4.28.2.3 INT EVENT_REQUEST::id

request id

De£nition at line 1232 of £le midas.h.

Referenced by bm_¤ush_cache(), bm_remove_event_request(), and bm_send_event().

4.28.2.4 INT EVENT_REQUEST::sampling_type

dispatch function

De£nition at line 1236 of £le midas.h.

Referenced by bm_¤ush_cache(), bm_remove_event_request(), and bm_send_event().

4.28.2.5 short int EVENT_REQUEST::trigger_mask

trigger mask

De£nition at line 1235 of £le midas.h.

Referenced by bm_¤ush_cache(), bm_push_event(), bm_receive_event(), and bm_-send_event().

4.28.2.6 BOOL EVENT_REQUEST::valid

indicating a valid entry

De£nition at line 1233 of £le midas.h.

Referenced by bm_¤ush_cache(), bm_push_event(), bm_receive_event(), bm_-remove_event_request(), and bm_send_event().

4.29 EXP_PARAM Struct Reference

4.29.1 Field Documentation

4.29.1.1 char EXP_PARAM::comment[80]

De£nition at line 27 of £le experim.h.

Referenced by ana_end_of_run().

4.30 FREE_DESCRIP Struct Reference

Data Fields

- INT size
- INT next_free

4.30.1 Field Documentation

4.30.1.1 INT FREE_DESCRIP::next_free

Address of next free block

De£nition at line 473 of £le msystem.h.

Referenced by db_open_database().

4.30.1.2 INT FREE_DESCRIP::size

size in bytes

De£nition at line 472 of £le msystem.h.

Referenced by db_open_database().

4.31 GLOBAL_PARAM Struct Reference

4.31.1 Field Documentation

4.31.1.1 ¤oat GLOBAL_PARAM::adc_threshold

De£nition at line 93 of £le experim.h.

4.32 HIST_RECORD Struct Reference

4.32.1 Field Documentation

4.32.1.1 DWORD HIST_RECORD::data_size

De£nition at line 1544 of £le midas.h.

4.32.1.2 DWORD HIST_RECORD::def_offset

De£nition at line 1543 of £le midas.h.

4.32.1.3 DWORD HIST_RECORD::event_id

De£nition at line 1541 of £le midas.h.

4.32.1.4 DWORD HIST_RECORD::record_type

De£nition at line 1540 of £le midas.h.

4.32.1.5 DWORD HIST_RECORD::time

De£nition at line 1542 of £le midas.h.

4.33 HISTORY Struct Reference

4.33.1 Field Documentation

4.33.1.1 DWORD HISTORY::base_time

De£nition at line 1567 of £le midas.h.

4.33.1.2 DWORD HISTORY::def_fh

De£nition at line 1566 of £le midas.h.

4.33.1.3 DWORD HISTORY::def_offset

De£nition at line 1568 of £le midas.h.

4.33.1.4 DWORD HISTORY::event_id

De£nition at line 1560 of £le midas.h.

4.33.1.5 char HISTORY::event_name[NAME_LENGTH]

De£nition at line 1561 of £le midas.h.

4.33.1.6 DWORD HISTORY::hist_fh

De£nition at line 1564 of £le midas.h.

4.33.1.7 DWORD HISTORY::index_fh

De£nition at line 1565 of £le midas.h.

4.33.1.8 DWORD HISTORY::n_tag

De£nition at line 1562 of £le midas.h.

4.33.1.9 TAG* HISTORY::tag

De£nition at line 1563 of £le midas.h.

4.34 INDEX_RECORD Struct Reference

4.34.1 Field Documentation

4.34.1.1 DWORD INDEX_RECORD::event_id

De£nition at line 1554 of £le midas.h.

4.34.1.2 DWORD INDEX_RECORD::offset

De£nition at line 1556 of £le midas.h.

4.34.1.3 DWORD INDEX_RECORD::time

De£nition at line 1555 of £le midas.h.

4.35 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

4.35.1 Field Documentation

4.35.1.1 WORD KEY::access_mode

Access mode

De£nition at line 1307 of £le midas.h.

Referenced by cm_cleanup(), db_create_key(), db_delete_key1(), db_£nd_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().

4.35.1.2 INT KEY::data

Address of variable (offset)

De£nition at line 1304 of £le midas.h.

Referenced by db_create_key(), db_delete_key1(), db_enum_key(), db_£nd_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().

4.35.1.3 INT KEY::item_size

Size of single data item

De£nition at line 1306 of £le 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().

4.35.1.4 INT KEY::last_written

Time of last write action

De£nition at line 1311 of £le midas.h.

Referenced by db_get_key_time(), db_set_data(), db_set_data_index(), and db_set_-value().

4.35.1.5 char KEY::name[NAME_LENGTH]

name of variable

De£nition at line 1303 of £le midas.h.

Referenced by cm_check_client(), cm_shutdown(), cm_transition(), db_check_-record(), db_copy(), db_create_key(), db_£nd_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().

4.35.1.6 INT KEY::next_key

Address of next key

De£nition at line 1309 of £le midas.h.

Referenced by db_create_key(), db_delete_key1(), db_enum_key(), and db_£nd_-key().

4.35.1.7 WORD KEY::notify_count

Notify counter

De£nition at line 1308 of £le midas.h.

Referenced by cm_cleanup(), db_delete_key1(), and db_open_database().

4.35.1.8 INT KEY::num_values

number of values

De£nition at line 1302 of £le 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().

4.35.1.9 INT KEY::parent_keylist

keylist to which this key belongs

De£nition at line 1310 of £le midas.h.

Referenced by db_create_key(), db_delete_key1(), db_enum_key(), db_£nd_key(), and db_open_database().

4.35.1.10 INT KEY::total_size

Total size of data block

De£nition at line 1305 of £le 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().

4.35.1.11 DWORD KEY::type

TID_xxx type

De£nition at line 1301 of £le midas.h.

Referenced by cm_transition(), db_check_record(), db_copy(), db_create_key(), db_-delete_key1(), db_enum_key(), db_£nd_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().

4.36 KEYLIST Struct Reference

Data Fields

- INT parent
- INT num_keys
- INT £rst_key

4.36.1 Field Documentation

4.36.1.1 INT KEYLIST::£rst_key

Address of £rst key

De£nition at line 1317 of £le midas.h.

Referenced by db_create_key(), db_delete_key1(), db_enum_key(), db_£nd_key(), and db_open_database().

4.36.1.2 INT KEYLIST::num_keys

number of keys

De£nition at line 1316 of £le midas.h.

Referenced by db_create_key(), db_delete_key1(), db_enum_key(), db_£nd_key(), db_get_key_info(), and db_open_database().

4.36.1.3 INT KEYLIST::parent

Address of parent key

De£nition at line 1315 of £le midas.h.

Referenced by db_create_key(), db_delete_key1(), db_enum_key(), db_£nd_key(), and db_open_database().

4.37 OPEN_RECORD Struct Reference

Data Fields

- INT handle
- WORD access_mode
- WORD ¤ags

4.37.1 Field Documentation

4.37.1.1 WORD OPEN_RECORD::access_mode

R/W ¤ags

De£nition at line 478 of £le msystem.h.

Referenced by cm_cleanup(), and db_open_database().

4.37.1.2 WORD OPEN_RECORD::¤ags

Data format, ...

De£nition at line 479 of £le msystem.h.

4.37.1.3 INT OPEN_RECORD::handle

Handle of record base key

De£nition at line 477 of £le msystem.h.

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

4.38 PROGRAM_INFO Struct Reference

4.38.1 Detailed Description

Program information structure

De£nition at line 1619 of £le midas.h.

4.38.2 Field Documentation

4.38.2.1 char PROGRAM_INFO::alarm_class[32]

De£nition at line 1627 of £le midas.h.

4.38.2.2 BOOL PROGRAM_INFO::auto_restart

De£nition at line 1626 of £le midas.h.

4.38.2.3 BOOL PROGRAM_INFO::auto_start

De£nition at line 1624 of £le midas.h.

Referenced by cm_transition().

4.38.2.4 BOOL PROGRAM_INFO::auto_stop

De£nition at line 1625 of £le midas.h.

Referenced by cm_transition().

4.38.2.5 DWORD PROGRAM_INFO::check_interval

De£nition at line 1622 of £le midas.h.

4.38.2.6 DWORD PROGRAM_INFO::£rst_failed

De£nition at line 1628 of £le midas.h.

4.38.2.7 BOOL PROGRAM_INFO::required

De£nition at line 1620 of £le midas.h.

4.38.2.8 char PROGRAM_INFO::start_command[256]

De£nition at line 1623 of £le midas.h.

Referenced by cm_transition().

4.38.2.9 INT PROGRAM_INFO::watchdog_timeout

De£nition at line 1621 of £le midas.h.

4.39 RECORD_LIST Struct Reference

4.39.1 Field Documentation

4.39.1.1 WORD RECORD_LIST::access_mode

De£nition at line 534 of £le msystem.h.

Referenced by db_close_all_records(), db_close_record(), db_open_record(), db_-send_changed_records(), and db_update_record().

4.39.1.2 INT RECORD_LIST::buf_size

De£nition at line 537 of £le msystem.h.

Referenced by db_open_record(), db_send_changed_records(), and db_update_-record().

4.39.1.3 void* RECORD_LIST::copy

De£nition at line 536 of £le msystem.h.

Referenced by db_open_record(), and db_send_changed_records().

4.39.1.4 void* RECORD_LIST::data

De£nition at line 535 of £le msystem.h.

Referenced by db_open_record(), db_send_changed_records(), and db_update_-record().

4.39.1.5 void(* RECORD_LIST::dispatcher)(INT, INT, void *)

Referenced by db_open_record(), and db_update_record().

4.39.1.6 HNDLE RECORD_LIST::handle

De£nition at line 532 of £le msystem.h.

Referenced by db_close_all_records(), db_close_record(), db_open_record(), and db_-update_record().

4.39.1.7 HNDLE RECORD_LIST::hDB

De£nition at line 533 of £le msystem.h.

Referenced by db_close_record(), and db_open_record().

4.39.1.8 void* RECORD_LIST::info

De£nition at line 539 of £le msystem.h.

Referenced by db_open_record().

4.40 REQUEST_LIST Struct Reference

4.40.1 Field Documentation

4.40.1.1 INT REQUEST_LIST::buffer_handle

De£nition at line 546 of £le msystem.h.

Referenced by bm_close_buffer(), bm_push_event(), and bm_request_event().

4.40.1.2 void(∗ REQUEST_LIST::dispatcher)(HNDLE, HNDLE, EVENT_HEADER ∗, void ∗)

Referenced by bm_push_event(), and bm_request_event().

4.40.1.3 short int REQUEST_LIST::event_id

De£nition at line 547 of £le msystem.h.

Referenced by bm_request_event().

4.40.1.4 short int REQUEST_LIST::trigger_mask

De£nition at line 548 of £le msystem.h.

Referenced by bm_request_event().

4.41 RUNINFO Struct Reference

4.41.1 Detailed Description

Contains the main parameters regarding the run status. The containt re¤ects the current system ONLY if Midas clients are connected. Otherwise the status is erroneous.

De£nition at line 1582 of £le 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

4.41.2 Field Documentation

4.41.2.1 INT RUNINFO::online_mode

Mode of operation online/of¤ine

De£nition at line 1584 of £le midas.h.

Referenced by ana_end_of_run().

4.41.2.2 INT RUNINFO::requested_transition

Deferred transition request

De£nition at line 1587 of £le midas.h.

4.41.2.3 INT RUNINFO::run_number

Current processing run number

De£nition at line 1585 of £le midas.h.

Referenced by ana_end_of_run().

4.41.2.4 char RUNINFO::start_time[32]

ASCII of the last start time

De£nition at line 1588 of £le midas.h.

Referenced by ana_end_of_run().

4.41.2.5 DWORD RUNINFO::start_time_binary

Bin of the last start time

De£nition at line 1589 of £le midas.h.

4.41.2.6 INT RUNINFO::state

Current run condition

De£nition at line 1583 of £le midas.h.

4.41.2.7 char RUNINFO::stop_time[32]

ASCII of the last stop time

De£nition at line 1590 of £le midas.h.

4.41.2.8 DWORD RUNINFO::stop_time_binary

ASCII of the last stop time

De£nition at line 1591 of £le midas.h.

4.41.2.9 INT RUNINFO::transition_in_progress

Intermediate state during transition

De£nition at line 1586 of £le midas.h.

4.42 SCALER_COMMON Struct Reference

4.42.1 Field Documentation

4.42.1.1 char SCALER_COMMON::buffer[32]

De£nition at line 181 of £le experim.h.

4.42.1.2 BOOL SCALER_COMMON::enabled

De£nition at line 185 of £le experim.h.

4.42.1.3 WORD SCALER_COMMON::event_id

De£nition at line 179 of £le experim.h.

4.42.1.4 double SCALER_COMMON::event_limit

De£nition at line 188 of £le experim.h.

4.42.1.5 char SCALER_COMMON::format[8]

De£nition at line 184 of £le experim.h.

4.42.1.6 char SCALER_COMMON::frontend_£le_name[256]

De£nition at line 193 of £le experim.h.

4.42.1.7 char SCALER_COMMON::frontend_host[32]

De£nition at line 191 of £le experim.h.

4.42.1.8 char SCALER_COMMON::frontend_name[32]

De£nition at line 192 of £le experim.h.

4.42.1.9 INT SCALER_COMMON::log_history

De£nition at line 190 of £le experim.h.

4.42.1.10 DWORD SCALER_COMMON::num_subevents

De£nition at line 189 of £le experim.h.

4.42.1.11 INT SCALER_COMMON::period

De£nition at line 187 of £le experim.h.

4.42.1.12 INT SCALER_COMMON::read_on

De£nition at line 186 of £le experim.h.

4.42.1.13 INT SCALER_COMMON::source

De£nition at line 183 of £le experim.h.

4.42.1.14 WORD SCALER_COMMON::trigger_mask

De£nition at line 180 of £le experim.h.

4.42.1.15 INT SCALER_COMMON::type

De£nition at line 182 of £le experim.h.

4.43 TAG Struct Reference

Data Fields

- char name [NAME_LENGTH]
- DWORD type
- DWORD n_data

4.43.1 Field Documentation

4.43.1.1 DWORD TAG::n_data

-

De£nition at line 1430 of £le midas.h.

4.43.1.2 char TAG::name[NAME_LENGTH]

-

De£nition at line 1428 of £le midas.h.

4.43.1.3 DWORD TAG::type

-

De£nition at line 1429 of £le midas.h.

4.44 TR_CLIENT Struct Reference

4.44.1 Detailed Description

dox**

De£nition at line 3583 of £le midas.c.

4.44.2 Field Documentation

4.44.2.1 char TR_CLIENT::client_name[NAME_LENGTH]

De£nition at line 3586 of £le midas.c.

4.44.2.2 char TR_CLIENT::host_name[HOST_NAME_LENGTH]

De£nition at line 3585 of £le midas.c.

4.44.2.3 int TR_CLIENT::port

De£nition at line 3587 of £le midas.c.

Referenced by cm_transition().

4.44.2.4 int TR_CLIENT::sequence_number

De£nition at line 3584 of £le midas.c.

Referenced by cm_transition().

4.45 TRIGGER_COMMON Struct Reference

4.45.1 Field Documentation

4.45.1.1 char TRIGGER_COMMON::buffer[32]

De£nition at line 125 of £le experim.h.

4.45.1.2 BOOL TRIGGER_COMMON::enabled

De£nition at line 129 of £le experim.h.

4.45.1.3 WORD TRIGGER_COMMON::event_id

De£nition at line 123 of £le experim.h.

4.45.1.4 double TRIGGER_COMMON::event_limit

De£nition at line 132 of £le experim.h.

4.45.1.5 char TRIGGER_COMMON::format[8]

De£nition at line 128 of £le experim.h.

4.45.1.6 char TRIGGER_COMMON::frontend_£le_name[256]

De£nition at line 137 of £le experim.h.

4.45.1.7 char TRIGGER_COMMON::frontend_host[32]

De£nition at line 135 of £le experim.h.

4.45.1.8 char TRIGGER_COMMON::frontend_name[32]

De£nition at line 136 of £le experim.h.

4.45.1.9 INT TRIGGER_COMMON::log_history

De£nition at line 134 of £le experim.h.

4.45.1.10 DWORD TRIGGER_COMMON::num_subevents

De£nition at line 133 of £le experim.h.

4.45.1.11 INT TRIGGER_COMMON::period

De£nition at line 131 of £le experim.h.

4.45.1.12 INT TRIGGER_COMMON::read_on

De£nition at line 130 of £le experim.h.

4.45.1.13 INT TRIGGER_COMMON::source

De£nition at line 127 of £le experim.h.

4.45.1.14 WORD TRIGGER_COMMON::trigger_mask

De£nition at line 124 of £le experim.h.

4.45.1.15 INT TRIGGER_COMMON::type

De£nition at line 126 of £le experim.h.

4.46 TRIGGER_SETTINGS Struct Reference

4.46.1 Field Documentation

4.46.1.1 BYTE TRIGGER_SETTINGS::io506

De£nition at line 163 of £le experim.h.

5 Midas File Documentation

5.1 adccalib.c File Reference

5.1.1 De£ne Documentation

5.1.1.1 #de£ne ADC_N_BINS 500

De£nition at line 96 of £le adccalib.c.

Referenced by adc_calib_init().

5.1.1.2 #de£ne ADC_X_HIGH 4000

De£nition at line 98 of £le adccalib.c.

Referenced by adc_calib_init().

5.1.1.3 #de£ne ADC_X_LOW 0

De£nition at line 97 of £le adccalib.c.

Referenced by adc_calib_init().

5.1.2 Function Documentation

5.1.2.1 INT adc_calib (EVENT_HEADER ∗, void ∗)

De£nition at line 135 of £le adccalib.c.

5.1.2.2 INT adc_calib_bor (INT *run_number*)

De£nition at line 121 of £le adccalib.c.

5.1.2.3 INT adc_calib_eor (INT *run_number*)

De£nition at line 128 of £le adccalib.c.

5.1.2.4 INT adc_calib_init (void)

De£nition at line 100 of £le 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,
}
```

De£nition at line 77 of £le adccalib.c.

5.1.3.2 ADC_CALIBRATION_PARAM adccalib_param

De£nition at line 64 of £le adccalib.c.

Referenced by adc_calib().

5.1.3.3 EXP_PARAM exp_param

De£nition at line 99 of £le analyzer.c.

Referenced by ana_end_of_run(), and analyzer_init().

5.1.3.4 TH1F∗ hAdcHists[N_ADC] `[static]`

De£nition at line 92 of £le adccalib.c.

Referenced by adc_calib(), and adc_calib_init().

5.1.3.5 RUNINFO runinfo

De£nition at line 97 of £le analyzer.c.

Referenced by ana_end_of_run(), and analyzer_init().

5.2 adcsum.c File Reference

5.2.1 De£ne Documentation

5.2.1.1 #de£ne DEFINE_TESTS

De£nition at line 57 of £le adcsum.c.

5.2.1.2 #de£ne PI 3.14159265359

De£nition at line 68 of £le adcsum.c.

5.2.2 Function Documentation

5.2.2.1 INT adc_summing (EVENT_HEADER ∗, void ∗)

De£nition at line 122 of £le adcsum.c.

5.2.2.2 INT adc_summing_bor (INT *run_number*)

5.2.2.3 INT adc_summing_init (void)

De£nition at line 107 of £le 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,
}
```

De£nition at line 88 of £le adcsum.c.

5.2.3.2 ADC_SUMMING_PARAM adc_summing_param

De£nition at line 73 of £le adcsum.c.

Referenced by adc_summing().

5.2.3.3 TH1F ∗ hAdcAvg `[static]`

De£nition at line 103 of £le adcsum.c.

Referenced by adc_summing(), and adc_summing_init().

5.2.3.4 TH1F∗ hAdcSum `[static]`

De£nition at line 103 of £le adcsum.c.

Referenced by adc_summing(), and adc_summing_init().

5.3 analyzer.c File Reference

5.3.1 Function Documentation

5.3.1.1 INT ana_begin_of_run (INT *run_number*, char ∗ *error*)

De£nition at line 247 of £le analyzer.c.

5.3.1.2 INT ana_end_of_run (INT *run_number*, char ∗ *error*)

De£nition at line 254 of £le analyzer.c.

5.3.1.3 INT ana_pause_run (INT *run_number*, char ∗ *error*)

De£nition at line 311 of £le analyzer.c.

5.3.1.4 INT ana_resume_run (INT *run_number*, char ∗ *error*)

De£nition at line 318 of £le analyzer.c.

5.3.1.5 INT analyzer_exit ()

De£nition at line 240 of £le analyzer.c.

5.3.1.6 INT analyzer_init ()

De£nition at line 185 of £le analyzer.c.

5.3.1.7 INT analyzer_loop ()

De£nition at line 325 of £le analyzer.c.

5.3.1.8 ASUM_BANK_STR (asum_bank_str)

5.3.2 Variable Documentation

5.3.2.1 ANA_MODULE adc_calib_module

De£nition at line 77 of £le adccalib.c.

5.3.2.2 ANA_MODULE adc_summing_module

De£nition at line 88 of £le adcsum.c.

5.3.2.3 BANK_LIST ana_scaler_bank_list[ ]

Initial value:

```
 {

   {"SCLR", TID_DWORD, N_ADC, NULL},


   {"ACUM", TID_DOUBLE, N_ADC, NULL},
   {""},
}
```

De£nition at line 136 of £le analyzer.c.

5.3.2.4 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},

   {""},
}
```

De£nition at line 123 of £le analyzer.c.

5.3.2.5 ANALYZE_REQUEST analyze_request[ ]

De£nition at line 147 of £le analyzer.c.

5.3.2.6 INT analyzer_loop_period = 0

De£nition at line 91 of £le analyzer.c.

5.3.2.7 char∗ analyzer_name = "Analyzer"

De£nition at line 88 of £le analyzer.c.

Referenced by analyzer_init().

5.3.2.8 EXP_PARAM exp_param

De£nition at line 99 of £le analyzer.c.

Referenced by ana_end_of_run(), and analyzer_init().

5.3.2.9 GLOBAL_PARAM global_param

De£nition at line 98 of £le analyzer.c.

5.3.2.10 INT odb_size = DEFAULT_ODB_SIZE

De£nition at line 94 of £le analyzer.c.

Referenced by cm_connect_experiment1().

5.3.2.11 RUNINFO runinfo

De£nition at line 97 of £le analyzer.c.

Referenced by ana_end_of_run(), and analyzer_init().

5.3.2.12 ANA_MODULE scaler_accum_module

De£nition at line 44 of £le scaler.c.

5.3.2.13 ANA_MODULE∗ scaler_module[ ]

Initial value:

```
 {
   &scaler_accum_module,
   NULL
}
```

De£nition at line 108 of £le analyzer.c.

5.3.2.14 ANA_MODULE∗ trigger_module[ ]

Initial value:

```
 {
   &adc_calib_module,
   &adc_summing_module,
   NULL
}
```

De£nition at line 113 of £le analyzer.c.

5.3.2.15 TRIGGER_SETTINGS trigger_settings

De£nition at line 100 of £le analyzer.c.

5.4 analyzer.dox File Reference

5.5 appendixA.dox File Reference

5.6 appendixB.dox File Reference

5.7 appendixC.dox File Reference

5.8 appendixD.dox File Reference

5.9 appendixE.dox File Reference

5.10 appendixG.dox File Reference

5.11 components.dox File Reference

5.12 ebuser.c File Reference

5.12.1 Detailed Description

The Event builder user £le

De£nition in £le 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

5.12.2 Function Documentation

5.12.2.1 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().

5.12.2.2 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().

5.12.2.3 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 £nal 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 ¤ag.

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

De£nition at line 222 of £le ebuser.c.

Referenced by source_scan().

5.12.2.4 INT ebuilder_exit ()

5.12.2.5 INT ebuilder_init ()

5.12.2.6 INT ebuilder_loop ()

5.12.2.7 INT ebuser (INT, BOOL *mismatch*, EBUILDER_CHANNEL ∗, EVENT_HEADER ∗, void ∗, INT ∗)

5.12.2.8 INT read_scaler_event (char ∗ *pevent*, INT *off*)

De£nition at line 403 of £le frontend.c.

5.12.3 Variable Documentation

5.12.3.1 BOOL debug

De£nition at line 292 of £le mfe.c.

5.12.3.2 INT display_period = 3000

De£nition at line 67 of £le ebuser.c.

5.12.3.3 EBUILDER_SETTINGS ebset

De£nition at line 80 of £le mevb.c.

Referenced by eb_user(), handFlush(), main(), source_booking(), source_scan(), and tr_start().

5.12.3.4 BOOL ebuilder_call_loop = FALSE

De£nition at line 64 of £le ebuser.c.

5.12.3.5 EQUIPMENT equipment[ ]

Initial value:

```
 {
   {"EB",
    {1, 0,
     "SYSTEM",
     0,
     0,
     "MIDAS",
     TRUE,
     },
    },

  {""}
}
```

De£nition at line 94 of £le ebuser.c.

5.12.3.6 INT event_buffer_size = 10 ∗ 10000

De£nition at line 76 of £le ebuser.c.

5.12.3.7 char∗ frontend_£le_name = __FILE__

De£nition at line 61 of £le ebuser.c.

5.12.3.8 char∗ frontend_name = "Ebuilder"

De£nition at line 58 of £le ebuser.c.

5.12.3.9 INT lModulo = 100

Global var for testing passed at BOR.

Globals

De£nition at line 80 of £le ebuser.c.

Referenced by eb_begin_of_run().

5.12.3.10 INT max_event_size = 10000

De£nition at line 70 of £le ebuser.c.

5.12.3.11 INT max_event_size_frag = 5 ∗ 1024 ∗ 1024

De£nition at line 73 of £le ebuser.c.

5.13 esone.c File Reference

5.13.1 Detailed Description

The ESONE CAMAC standard call £le

De£nition in £le 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[ ])

5.13.2 Function Documentation

5.13.2.1 INLINE void cccc (const int *ext*)

Control Crate Clear.

Generate Crate Clear function. Execute cam_crate_clear()

Parameters:

ext external address

Returns:

void

De£nition at line 202 of £le esone.c.

5.13.2.2 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

De£nition at line 276 of £le esone.c.

5.13.2.3 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

De£nition at line 237 of £le esone.c.

5.13.2.4 INLINE void cccz (const int *ext*)

Control Crate Z.

Generate Dataway Initialize. Execute cam_crate_zinit()

Parameters:

ext external address

Returns:

void

De£nition at line 219 of £le esone.c.

5.13.2.5 INLINE void ccinit (void)

CAMAC initialization

CAMAC initialization must be called before any other ESONE subroutine call.

Returns:

void

De£nition at line 81 of £le esone.c.

5.13.2.6 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

De£nition at line 437 of £le esone.c.

5.13.2.7 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

De£nition at line 357 of £le esone.c.

5.13.2.8 INLINE void cclnk (const int *lam*, void(*)(void) *isr*)

Link LAM to service procedure

Link a speci£c 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

De£nition at line 380 of £le esone.c.

5.13.2.9 INLINE void ccrgl (const int *lam*)

Relink LAM

Re-enable LAM in the controller

Parameters:

lam external address

Returns:

void

De£nition at line 417 of £le esone.c.

5.13.2.10 INLINE void cculk (const int *lam*)

Unlink LAM from service procedure

Performs complementary operation to cclnk.

Parameters:

lam external address

Returns:

void

De£nition at line 400 of £le esone.c.

5.13.2.11 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

De£nition at line 320 of £le esone.c.

5.13.2.12 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

De£nition at line 118 of £le esone.c.

Referenced by cdlam().

5.13.2.13 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

De£nition at line 476 of £le esone.c.

5.13.2.14 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] : £rst 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

De£nition at line 530 of £le esone.c.

5.13.2.15 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

De£nition at line 174 of £le esone.c.

Referenced by cfga(), cfubc(), and cfubr().

5.13.2.16 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

De£nition at line 627 of £le esone.c.

5.13.2.17 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

De£nition at line 690 of £le esone.c.

5.13.2.18 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

De£nition at line 499 of £le esone.c.

5.13.2.19 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] : £rst 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

De£nition at line 582 of £le esone.c.

5.13.2.20 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

De£nition at line 139 of £le esone.c.

Referenced by csga(), csubc(), and csubr().

5.13.2.21 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

De£nition at line 658 of £le esone.c.

5.13.2.22 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

De£nition at line 721 of £le esone.c.

5.13.2.23 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

De£nition at line 298 of £le esone.c.

5.13.2.24 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

De£nition at line 258 of £le esone.c.

5.13.2.25 INLINE void ctgl (const int *ext*, int * *l*)

Control Test Demand Present.

Test the LAM register.

Parameters:

- ***ext*** external LAM register address
- ***l*** l !=0 if any LAM is set.

Returns:

void

De£nition at line 337 of £le esone.c.

5.13.2.26 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

De£nition at line 455 of £le esone.c.

5.13.2.27 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

De£nition at line 95 of £le esone.c.

5.14 eventbuilder.dox File Reference

5.15 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

5.15.1 De£ne Documentation

5.15.1.1 #de£ne ADC_CALIBRATION_PARAM_DEFINED

De£nition at line 38 of £le experim.h.

5.15.1.2 #de£ne 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 }
```

De£nition at line 46 of £le experim.h.

5.15.1.3 #de£ne ADC_SUMMING_PARAM_DEFINED

De£nition at line 74 of £le experim.h.

5.15.1.4 #de£ne ADC_SUMMING_PARAM_STR(_name)

Value:

```
char *_name[] = {\
"[.]",\
"ADC threshold = FLOAT : 5",\
"",\
NULL }
```

De£nition at line 80 of £le experim.h.

5.15.1.5 #de£ne ASUM_BANK_DEFINED

De£nition at line 106 of £le experim.h.

5.15.1.6 #de£ne ASUM_BANK_STR(_name)

Value:

```
char *_name[] = {\
"[.]",\
"Sum = FLOAT : 0",\
"Average = FLOAT : 0",\
"",\
NULL }
```

De£nition at line 113 of £le experim.h.

5.15.1.7 #de£ne EXP_PARAM_DEFINED

De£nition at line 24 of £le experim.h.

5.15.1.8 #de£ne EXP_PARAM_STR(_name)

Value:

```
char *_name[] = {\
"[.]",\
"Comment = STRING : [80] Test",\
"",\
NULL }
```

De£nition at line 30 of £le experim.h.

Referenced by analyzer_init().

5.15.1.9 #de£ne GLOBAL_PARAM_DEFINED

De£nition at line 90 of £le experim.h.

5.15.1.10 #de£ne GLOBAL_PARAM_STR(_name)

Value:

```
char *_name[] = {\
"[.]",\
"ADC Threshold = FLOAT : 5",\
"",\
NULL }
```

De£nition at line 96 of £le experim.h.

Referenced by analyzer_init().

5.15.1.11 #de£ne SCALER_COMMON_DEFINED

De£nition at line 176 of £le experim.h.

5.15.1.12 #de£ne 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 }
```

De£nition at line 196 of £le experim.h.

5.15.1.13 #de£ne TRIGGER_COMMON_DEFINED

De£nition at line 120 of £le experim.h.

5.15.1.14 #de£ne 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 }
```

De£nition at line 140 of £le experim.h.

5.15.1.15 #de£ne TRIGGER_SETTINGS_DEFINED

De£nition at line 160 of £le experim.h.

5.15.1.16 #de£ne TRIGGER_SETTINGS_STR(_name)

Value:

```
char *_name[] = {\
"[.]",\
"IO506 = BYTE : 7",\
"",\
NULL }
```

De£nition at line 166 of £le experim.h.

Referenced by analyzer_init().

5.16 frontend.c File Reference

5.16.1 De£ne Documentation

5.16.1.1 #de£ne CRATE 0

De£nition at line 119 of £le frontend.c.

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

5.16.1.2 #de£ne N_ADC 4

De£nition at line 114 of £le frontend.c.

Referenced by adc_calib().

5.16.1.3 #de£ne N_SCLR 4

De£nition at line 116 of £le frontend.c.

5.16.1.4 #de£ne N_TDC 4

De£nition at line 115 of £le frontend.c.

5.16.1.5 #de£ne SLOT_ADC 1

De£nition at line 121 of £le frontend.c.

Referenced by read_trigger_event().

5.16.1.6 #de£ne SLOT_IO 23

De£nition at line 120 of £le frontend.c.

Referenced by frontend_init(), and read_trigger_event().

5.16.1.7 #de£ne SLOT_SCLR 3

De£nition at line 123 of £le frontend.c.

Referenced by read_scaler_event().

5.16.1.8 #de£ne SLOT_TDC 2

De£nition at line 122 of £le frontend.c.

Referenced by read_trigger_event().

5.16.2 Function Documentation

5.16.2.1 INT begin_of_run (INT *run_number*, char ∗ *error*)

Referenced by tr_start().

5.16.2.2 INT end_of_run (INT *run_number*, char ∗ *error*)

Referenced by tr_stop().

5.16.2.3 INT frontend_exit ()

5.16.2.4 INT frontend_init ()

5.16.2.5 INT frontend_loop ()

5.16.2.6 INT interrupt_con£gure (INT *cmd*, INT *source*, PTYPE *adr*)

De£nition at line 316 of £le frontend.c.

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

5.16.2.7 INT pause_run (INT *run_number*, char ∗ *error*)

Referenced by tr_pause().

5.16.2.8 INT poll_event (INT *source*, INT *count*, BOOL *test*)

De£nition at line 295 of £le frontend.c.

Referenced by register_equipment(), and scheduler().

5.16.2.9 INT read_scaler_event (char ∗ *pevent*, INT *off*)

5.16.2.10 INT read_trigger_event (char ∗ *pevent*, INT *off*)

De£nition at line 333 of £le frontend.c.

5.16.2.11 INT resume_run (INT *run_number*, char ∗ *error*)

Referenced by tr_resume().

5.16.3 Variable Documentation

5.16.3.1 INT display_period = 3000

De£nition at line 102 of £le frontend.c.

5.16.3.2 EQUIPMENT equipment[ ]

De£nition at line 142 of £le frontend.c.

5.16.3.3 INT event_buffer_size = 10 ∗ 10000

De£nition at line 111 of £le frontend.c.

5.16.3.4 BOOL frontend_call_loop = FALSE

De£nition at line 99 of £le frontend.c.

5.16.3.5 char∗ frontend_£le_name = __FILE__

De£nition at line 96 of £le frontend.c.

5.16.3.6 char∗ frontend_name = "Sample Frontend"

De£nition at line 94 of £le frontend.c.

5.16.3.7 INT max_event_size = 10000

De£nition at line 105 of £le frontend.c.

5.16.3.8 INT max_event_size_frag = 5 ∗ 1024 ∗ 1024

De£nition at line 108 of £le frontend.c.

5.17 internal.dox File Reference

5.18 introduction.dox File Reference

5.19 mcstd.h File Reference

5.19.1 Detailed Description

The Midas CAMAC include £le

De£nition in £le 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 INLINE void EXPRT 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)

5.20 mevb.c File Reference

De£nes

- #de£ne 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 ∗)

5.20.1 De£ne Documentation

5.20.1.1 #de£ne DEFAULT_FE_TIMEOUT 60000

De£nition at line 78 of £le mevb.c.

5.20.1.2 #de£ne 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 £le name = STRING : [256] \n\"

De£nition at line 133 of £le mevb.c.

5.20.1.3 #de£ne EQUIPMENT_STATISTICS_STR "\Events sent = DOUBLE : 0\n\Events per sec. = DOUBLE : 0\n\kBytes per sec. = DOUBLE : 0\n\"

De£nition at line 151 of £le mevb.c.

5.20.1.4 #de£ne ODB_UPDATE_TIME 1000

De£nition at line 76 of £le mevb.c.

5.20.1.5 #de£ne SERVER_CACHE_SIZE 100000

dox∗∗

De£nition at line 74 of £le mevb.c.

5.20.2 Function Documentation

5.20.2.1 INT close_buffers (void)

De£nition at line 856 of £le mevb.c.

Referenced by scan_fragment().

5.20.2.2 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

De£nition at line 140 of £le ebuser.c.

5.20.2.3 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

De£nition at line 155 of £le ebuser.c.

5.20.2.4 INT eb_mfragment_add (char ∗ *pdest*, char ∗ *psrce*, INT ∗ *size*)

De£nition at line 458 of £le mevb.c.

5.20.2.5 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 £nal 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 ¤ag.

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

De£nition at line 222 of £le ebuser.c.

Referenced by source_scan().

5.20.2.6 INT eb_yfragment_add (char ∗ *pdest*, char ∗ *psrce*, INT ∗ *size*)

De£nition at line 514 of £le mevb.c.

5.20.2.7 INT ebuilder_exit (void)

De£nition at line 121 of £le ebuser.c.

Referenced by main().

5.20.2.8 INT ebuilder_init (void)

De£nition at line 115 of £le ebuser.c.

Referenced by main().

5.20.2.9 INT ebuilder_loop (void)

De£nition at line 127 of £le ebuser.c.

5.20.2.10 void free_event_buffer (INT *nfrag*)

De£nition at line 704 of £le mevb.c.

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

5.20.2.11 INT handFlush (void)

De£nition at line 716 of £le mevb.c.

Referenced by close_buffers().

5.20.2.12 INT load_fragment (void)

De£nition at line 285 of £le mevb.c.

Referenced by main().

5.20.2.13 int main (unsigned int *argc*, char ∗∗ *argv*)

De£nition at line 1065 of £le mevb.c.

5.20.2.14 INT register_equipment (void)

De£nition at line 158 of £le mevb.c.

5.20.2.15 INT scan_fragment (void)

De£nition at line 358 of £le mevb.c.

Referenced by main().

5.20.2.16 INT source_booking (void)

De£nition at line 750 of £le mevb.c.

Referenced by tr_start().

5.20.2.17 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)

De£nition at line 905 of £le mevb.c.

Referenced by scan_fragment().

5.20.2.18 INT source_unbooking (void)

De£nition at line 820 of £le mevb.c.

Referenced by close_buffers(), and main().

5.20.2.19 INT tr_start (INT *rn*, char ∗ *error*)

De£nition at line 586 of £le mevb.c.

5.20.2.20 INT tr_stop (INT *rn*, char ∗ *error*)

De£nition at line 691 of £le mevb.c.

5.20.2.21 INT ybos_event_swap (DWORD ∗ *pevt*)

Referenced by eb_yfragment_add(), and source_scan().

5.20.3 Variable Documentation

5.20.3.1 BOOL abort_requested = FALSE stop_requested = TRUE

De£nition at line 101 of £le mevb.c.

Referenced by close_buffers(), scan_fragment(), and tr_start().

5.20.3.2 DWORD actual_millitime

De£nition at line 87 of £le mevb.c.

5.20.3.3 DWORD actual_time

De£nition at line 86 of £le mevb.c.

5.20.3.4 char bars[ ] = "|\\-/"

De£nition at line 99 of £le mevb.c.

Referenced by scan_fragment(), and source_scan().

5.20.3.5 char buffer_name[NAME_LENGTH]

De£nition at line 92 of £le mevb.c.

Referenced by bm_open_buffer(), bm_push_event(), load_fragment(), and main().

5.20.3.6 BOOL debug = FALSE debug1 = FALSE

De£nition at line 96 of £le mevb.c.

5.20.3.7 char* dest_event

De£nition at line 94 of £le mevb.c.

Referenced by load_fragment(), and source_scan().

5.20.3.8 INT display_period

De£nition at line 67 of £le ebuser.c.

5.20.3.9 EBUILDER_CHANNEL ebch[MAX_CHANNELS]

De£nition at line 81 of £le mevb.c.

Referenced by eb_user(), free_event_buffer(), handFlush(), load_fragment(), main(), scan_fragment(), source_booking(), source_scan(), and source_unbooking().

5.20.3.10 EBUILDER_SETTINGS ebset

De£nition at line 80 of £le mevb.c.

Referenced by eb_user(), handFlush(), main(), source_booking(), source_scan(), and tr_start().

5.20.3.11 EQUIPMENT equipment[ ]

De£nition at line 94 of £le ebuser.c.

5.20.3.12 INT event_buffer_size

De£nition at line 76 of £le ebuser.c.

5.20.3.13 char expt_name[NAME_LENGTH]

De£nition at line 90 of £le mevb.c.

Referenced by main().

5.20.3.14 BOOL frontend_call_loop

De£nition at line 99 of £le frontend.c.

5.20.3.15 char* frontend_£le_name

De£nition at line 61 of £le ebuser.c.

5.20.3.16 char* frontend_name

De£nition at line 58 of £le ebuser.c.

5.20.3.17 char full_frontend_name[256]

De£nition at line 91 of £le mevb.c.

5.20.3.18 HNDLE hDB

De£nition at line 95 of £le mevb.c.

5.20.3.19 HNDLE hEqKey

De£nition at line 95 of £le mevb.c.

Referenced by load_fragment().

5.20.3.20 HNDLE hESetKey

De£nition at line 95 of £le mevb.c.

5.20.3.21 HNDLE hKey

De£nition at line 95 of £le mevb.c.

Referenced by analyzer_init(), cm_connect_client(), cm_delete_client_info(), 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_create_key(), db_create_link(), db_create_record(), db_delete_key(), db_delete_key1(), db_enum_key(), db_£nd_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_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(), logger_root(), main(), register_equipment(), tr_-start(), and update_odb().

5.20.3.22 char host_name[HOST_NAME_LENGTH]

De£nition at line 89 of £le mevb.c.

5.20.3.23 HNDLE hStatKey

De£nition at line 95 of £le mevb.c.

5.20.3.24 HNDLE hSubkey

De£nition at line 95 of £le 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().

5.20.3.25 int i_bar

De£nition at line 100 of £le mevb.c.

Referenced by scan_fragment().

5.20.3.26 DWORD last_time

De£nition at line 85 of £le mevb.c.

Referenced by scan_fragment().

5.20.3.27 INT max_event_size

De£nition at line 70 of £le ebuser.c.

5.20.3.28 INT max_event_size_frag

De£nition at line 73 of £le ebuser.c.

5.20.3.29 INT(∗ meb_fragment_add)(char ∗, char ∗, INT ∗)

De£nition at line 104 of £le mevb.c.

Referenced by load_fragment(), and source_scan().

5.20.3.30 INT nfragment

De£nition at line 93 of £le mevb.c.

Referenced by load_fragment(), source_booking(), source_scan(), and source_-unbooking().

5.20.3.31 INT run_number

De£nition at line 84 of £le mevb.c.

5.20.3.32 INT run_state

De£nition at line 83 of £le mevb.c.

5.20.3.33 DWORD stop_time = 0 request_stop_time = 0

De£nition at line 102 of £le mevb.c.

Referenced by close_buffers().

5.20.3.34 BOOL wheel = FALSE

De£nition at line 98 of £le mevb.c.

Referenced by main().

5.21 mfe.c File Reference

5.21.1 De£ne Documentation

5.21.1.1 #de£ne DEFAULT_FE_TIMEOUT 60000

De£nition at line 278 of £le mfe.c.

Referenced by main().

5.21.1.2 #de£ne 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 £le name = STRING : [256] \n\"

De£nition at line 330 of £le mfe.c.

Referenced by register_equipment().

5.21.1.3 #de£ne EQUIPMENT_STATISTICS_STR "\Events sent = DOUBLE : 0\n\Events per sec. = DOUBLE : 0\n\kBytes per sec. = DOUBLE : 0\n\"

De£nition at line 348 of £le mfe.c.

Referenced by register_equipment().

5.21.1.4 #de£ne ODB_UPDATE_TIME 1000

De£nition at line 276 of £le mfe.c.

Referenced by scheduler().

5.21.1.5 #de£ne SERVER_CACHE_SIZE 100000

De£nition at line 274 of £le mfe.c.

Referenced by register_equipment().

5.21.2 Function Documentation

5.21.2.1 INT begin_of_run (INT *run_number*, char ∗ *error*)

De£nition at line 248 of £le frontend.c.

5.21.2.2 void display (BOOL *bInit*)

De£nition at line 1232 of £le mfe.c.

Referenced by main(), and scheduler().

5.21.2.3 INT end_of_run (INT *run_number*, char ∗ *error*)

De£nition at line 257 of £le frontend.c.

5.21.2.4 INT frontend_exit (void)

De£nition at line 241 of £le frontend.c.

Referenced by main().

5.21.2.5 INT frontend_init (void)

De£nition at line 216 of £le frontend.c.

Referenced by main().

5.21.2.6 INT frontend_loop (void)

De£nition at line 278 of £le frontend.c.

Referenced by scheduler().

5.21.2.7 INT get_frontend_index ()

De£nition at line 1896 of £le mfe.c.

5.21.2.8 INT interrupt_con£gure (INT *cmd*, INT *source*, PTYPE *adr*)

De£nition at line 316 of £le frontend.c.

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

5.21.2.9 void interrupt_enable (BOOL ¤*ag*)

De£nition at line 1149 of £le mfe.c.

Referenced by main(), scheduler(), tr_pause(), tr_resume(), tr_start(), and tr_stop().

5.21.2.10 void interrupt_routine (void)

De£nition at line 1163 of £le mfe.c.

Referenced by register_equipment().

5.21.2.11 BOOL logger_root ()

De£nition at line 1303 of £le mfe.c.

Referenced by scheduler().

5.21.2.12 int main (int *argc*, char ∗ *argv*[ ])

De£nition at line 1906 of £le mfe.c.

5.21.2.13 INT manual_trigger (INT *index*, void ∗ *prpc_param*[ ])

De£nition at line 490 of £le mfe.c.

Referenced by register_equipment().

5.21.2.14 int message_print (const char ∗ *msg*)

De£nition at line 1214 of £le mfe.c.

Referenced by main().

5.21.2.15 INT pause_run (INT *run_number*, char ∗ *error*)

De£nition at line 264 of £le frontend.c.

5.21.2.16 INT poll_event (INT *source*, INT *count*, BOOL *test*)

De£nition at line 295 of £le frontend.c.

Referenced by register_equipment(), and scheduler().

5.21.2.17 INT register_equipment (void)

De£nition at line 498 of £le mfe.c.

Referenced by main().

5.21.2.18 INT resume_run (INT *run_number*, char ∗ *error*)

De£nition at line 271 of £le frontend.c.

5.21.2.19 INT scheduler (void)

De£nition at line 1330 of £le mfe.c.

Referenced by main().

5.21.2.20 void send_all_periodic_events (INT *transition*)

De£nition at line 1121 of £le mfe.c.

Referenced by tr_pause(), tr_resume(), tr_start(), and tr_stop().

5.21.2.21 int send_event (INT *index*)

De£nition at line 928 of £le mfe.c.

Referenced by scheduler(), and send_all_periodic_events().

5.21.2.22 INT tr_pause (INT *rn*, char ∗ *error*)

De£nition at line 441 of £le mfe.c.

Referenced by main().

5.21.2.23 INT tr_resume (INT *rn*, char ∗ *error*)

De£nition at line 466 of £le mfe.c.

Referenced by main().

5.21.2.24 INT tr_start (INT *rn*, char ∗ *error*)

De£nition at line 358 of £le mfe.c.

Referenced by main().

5.21.2.25 INT tr_stop (INT *rn*, char ∗ *error*)

De£nition at line 392 of £le mfe.c.

Referenced by main().

5.21.2.26 void update_odb (EVENT_HEADER ∗ *pevent*, HNDLE *hKey*, INT *format*)

De£nition at line 782 of £le mfe.c.

Referenced by scheduler(), and send_event().

5.21.3 Variable Documentation

5.21.3.1 DWORD actual_millitime

De£nition at line 283 of £le mfe.c.

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

5.21.3.2 DWORD actual_time

De£nition at line 282 of £le mfe.c.

Referenced by scheduler().

5.21.3.3 DWORD auto_restart = 0

De£nition at line 293 of £le mfe.c.

Referenced by scheduler().

5.21.3.4 BOOL debug

De£nition at line 292 of £le mfe.c.

Referenced by main().

5.21.3.5 INT display_period

De£nition at line 67 of £le ebuser.c.

Referenced by scheduler().

5.21.3.6 EQUIPMENT equipment[ ]

De£nition at line 94 of £le 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().

5.21.3.7 INT event_buffer_size

De£nition at line 76 of £le ebuser.c.

Referenced by main().

5.21.3.8 char exp_name[NAME_LENGTH]

Definition at line 286 of file mfe.c.

Referenced by cm_connect_experiment(), cm_connect_experiment1(), cm_get_-environment(), cm_list_experiments(), cm_select_experiment(), and main().

5.21.3.9 INT fe_stop = 0

Definition at line 291 of file mfe.c.

5.21.3.10 BOOL frontend_call_loop

Definition at line 99 of file frontend.c.

5.21.3.11 char* frontend_file_name

Definition at line 61 of file ebuser.c.

Referenced by register_equipment().

5.21.3.12 INT frontend_index = -1

Definition at line 295 of file mfe.c.

Referenced by main(), and register_equipment().

5.21.3.13 char* frontend_name

Definition at line 58 of file ebuser.c.

Referenced by main().

5.21.3.14 char full_frontend_name[256]

Definition at line 287 of file mfe.c.

Referenced by display(), load_fragment(), main(), register_equipment(), scan_-fragment(), source_scan(), tr_start(), and tr_stop().

5.21.3.15 HNDLE hDB

Definition at line 297 of file mfe.c.

Referenced by al_trigger_alarm(), ana_end_of_run(), analyzer_init(), cm_check_-client(), cm_connect_client(), cm_connect_experiment1(), cm_delete_client_-info(), 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_create_key(), db_create_link(), db_create_record(), db_delete_key(), db_delete_key1(), db_enum_key(), db_£nd_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_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_unlock_-database(), db_update_record(), el_submit(), load_fragment(), logger_root(), main(), register_equipment(), scheduler(), tr_start(), and update_odb().

5.21.3.16 char host_name[HOST_NAME_LENGTH]

De£nition at line 285 of £le 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().

5.21.3.17 BOOL interrupt_enabled

De£nition at line 1147 of £le mfe.c.

Referenced by interrupt_enable().

5.21.3.18 EQUIPMENT* interrupt_eq = NULL

De£nition at line 318 of £le mfe.c.

Referenced by interrupt_routine(), main(), register_equipment(), and scheduler().

5.21.3.19 EVENT_HEADER* interrupt_odb_buffer

De£nition at line 319 of £le mfe.c.

Referenced by interrupt_routine(), main(), register_equipment(), and scheduler().

5.21.3.20 BOOL interrupt_odb_buffer_valid

De£nition at line 320 of £le mfe.c.

Referenced by interrupt_routine(), and scheduler().

5.21.3.21 INT manual_trigger_event_id = 0

De£nition at line 294 of £le mfe.c.

Referenced by manual_trigger(), and scheduler().

5.21.3.22 INT max_bytes_per_sec

De£nition at line 289 of £le mfe.c.

Referenced by scheduler().

5.21.3.23 INT max_event_size

De£nition at line 70 of £le ebuser.c.

Referenced by load_fragment(), main(), scheduler(), send_event(), and source_-booking().

5.21.3.24 INT max_event_size_frag

De£nition at line 73 of £le ebuser.c.

Referenced by main(), and send_event().

5.21.3.25 INT optimize = 0

De£nition at line 290 of £le mfe.c.

5.21.3.26 INT run_number

De£nition at line 281 of £le mfe.c.

Referenced by close_buffers(), cm_transition(), display(), el_submit(), register_-equipment(), scheduler(), tr_pause(), tr_resume(), tr_start(), and tr_stop().

5.21.3.27 INT run_state

De£nition at line 280 of £le mfe.c.

Referenced by close_buffers(), display(), handFlush(), main(), register_equipment(), scheduler(), tr_pause(), tr_resume(), tr_start(), and tr_stop().

5.22 mhttpd.dox File Reference

5.23 midas.c File Reference

5.23.1 Detailed Description

The main core C-code for Midas.

De£nition in £le 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, char ∗£lename, INT line, const char ∗routine, const char ∗format,...)
- INT cm_msg1 (INT message_type, char ∗£lename, 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 ∗time)
- char ∗ cm_get_version ()
- 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 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 ∗perror, INT strsize, INT async_¤ag, INT debug_¤ag)
- 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_¤ag)
- INT bm_¤ush_cache (INT buffer_handle, INT async_¤ag)

- INT bm_receive_event (INT buffer_handle, void *destination, INT *buf_size, INT async_¤ag)
- 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_¤ag)
- INT rpc_¤ush_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_£nd (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 hs_set_path (char *path)
- INT hs_open_£le (DWORD ltime, char *suf£x, INT mode, int *fh)
- INT el_submit (int run, char *author, char *type, char *system, char *subject, char *text, char *reply_to, char *encoding, char *a£lename1, char *buffer1, INT buffer_size1, char *a£lename2, char *buffer2, INT buffer_size2, char *a£lename3, char *buffer3, INT buffer_size3, char *tag, INT tag_size)
- INT al_trigger_alarm (char *alarm_name, char *alarm_message, char *default_-class, char *cond_str, INT type)
- INT dm_buffer_create (INT size, INT user_max_event_size)

Variables

- HNDLE _hKeyClient = 0

5.24 midas.dox File Reference

5.25 midas.h File Reference

5.25.1 Detailed Description

The main include £le

De£nition in £le 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 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_TEST
- struct HIST_RECORD
- struct DEF_RECORD
- struct INDEX_RECORD
- struct HISTORY
- struct RUNINFO
- struct PROGRAM_INFO
- struct ALARM_CLASS
- struct ALARM

Defines

- #define TAPE_BUFFER_SIZE 0x8000
- #define NET_TCP_SIZE 0xFFFF
- #define OPT_TCP_SIZE 8192
- #define NET_UDP_SIZE 8192
- #define EVENT_BUFFER_SIZE 0x100000
- #define EVENT_BUFFER_NAME "SYSTEM"
- #define MAX_EVENT_SIZE 0x80000
- #define DEFAULT_EVENT_BUFFER_SIZE 0x200000;
- #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

- #de£ne TID_FLOAT 9
- #de£ne TID_DOUBLE 10
- #de£ne TID_BITFIELD 11
- #de£ne TID_STRING 12
- #de£ne TID_ARRAY 13
- #de£ne TID_STRUCT 14
- #de£ne TID_KEY 15
- #de£ne TID_LINK 16
- #de£ne TID_LAST 17
- #de£ne SYNC 0
- #de£ne MODE_READ (1<<0)
- #de£ne RPC_OTIMEOUT 1
- #de£ne WF_WATCH_ME (1<<0)
- #de£ne TR_START (1<<0)
- #de£ne TR_STOP (1<<1)
- #de£ne TR_PAUSE (1<<2)
- #de£ne TR_RESUME (1<<3)
- #de£ne EQ_PERIODIC (1<<0)
- #de£ne EQ_POLLED (1<<1)
- #de£ne EQ_INTERRUPT (1<<2)
- #de£ne EQ_SLOW (1<<3)
- #de£ne EQ_MANUAL_TRIG (1<<4)
- #de£ne EQ_FRAGMENTED (1<<5)
- #de£ne EQ_EB (1<<6)
- #de£ne RO_RUNNING (1<<0)
- #de£ne RO_STOPPED (1<<1)
- #de£ne RO_PAUSED (1<<2)
- #de£ne RO_BOR (1<<3)
- #de£ne RO_EOR (1<<4)
- #de£ne RO_PAUSE (1<<5)
- #de£ne RO_RESUME (1<<6)
- #de£ne RO_TRANSITIONS (RO_BOR|RO_EOR|RO_PAUSE|RO_RESUME)
- #de£ne RO_ALWAYS (0xFF)
- #de£ne RO_ODB (1<<8)
- #de£ne CH_BS 8
- #de£ne LAM_SOURCE(c, s) (c<<24 | ((s) & 0xFFFFFF))
- #de£ne LAM_STATION(s) (1<<(s-1))
- #de£ne LAM_SOURCE_CRATE(c) (c>>24)
- #de£ne LAM_SOURCE_STATION(s) ((s) & 0xFFFFFF)
- #de£ne CNAF 0x1
- #de£ne MAX(a, b) (((a) > (b)) ? (a) : (b))
- #de£ne MIN(a, b) (((a) < (b)) ? (a) : (b))
- #de£ne ALIGN8(x) (((x)+7) & ~7)

- #de£ne VALIGN(adr, align) (((PTYPE) (adr)+align-1) & ∼(align-1))
- #de£ne MT_ERROR (1<<0)
- #de£ne MT_INFO (1<<1)
- #de£ne MT_DEBUG (1<<2)
- #de£ne MT_USER (1<<3)
- #de£ne MT_LOG (1<<4)
- #de£ne MT_TALK (1<<5)
- #de£ne MT_CALL (1<<6)
- #de£ne MT_ALL 0xFF
- #de£ne MERROR MT_ERROR, __FILE__, __LINE__
- #de£ne MINFO MT_INFO, __FILE__, __LINE__
- #de£ne MDEBUG MT_DEBUG, __FILE__, __LINE__
- #de£ne MUSER MT_USER, __FILE__, __LINE__
- #de£ne MLOG MT_LOG, __FILE__, __LINE__
- #de£ne MTALK MT_TALK, __FILE__, __LINE__
- #de£ne MCALL MT_CALL, __FILE__, __LINE__
- #de£ne SUCCESS 1
- #de£ne CM_SUCCESS 1
- #de£ne CM_SET_ERROR 102
- #de£ne CM_NO_CLIENT 103
- #de£ne CM_DB_ERROR 104
- #de£ne CM_UNDEF_EXP 105
- #de£ne CM_VERSION_MISMATCH 106
- #de£ne CM_SHUTDOWN 107
- #de£ne CM_WRONG_PASSWORD 108
- #de£ne CM_UNDEF_ENVIRON 109
- #de£ne CM_DEFERRED_TRANSITION 110
- #de£ne CM_TRANSITION_IN_PROGRESS 111
- #de£ne CM_TIMEOUT 112
- #de£ne CM_INVALID_TRANSITION 113
- #de£ne CM_TOO_MANY_REQUESTS 114
- #de£ne BM_SUCCESS 1
- #de£ne BM_CREATED 202
- #de£ne BM_NO_MEMORY 203
- #de£ne BM_INVALID_NAME 204
- #de£ne BM_INVALID_HANDLE 205
- #de£ne BM_NO_SLOT 206
- #de£ne BM_NO_MUTEX 207
- #de£ne BM_NOT_FOUND 208
- #de£ne BM_ASYNC_RETURN 209
- #de£ne BM_TRUNCATED 210
- #de£ne BM_MULTIPLE_HOSTS 211
- #de£ne BM_MEMSIZE_MISMATCH 212

- #de£ne BM_CONFLICT 213
- #de£ne BM_EXIT 214
- #de£ne BM_INVALID_PARAM 215
- #de£ne BM_MORE_EVENTS 216
- #de£ne BM_INVALID_MIXING 217
- #de£ne BM_NO_SHM 218
- #de£ne DB_SUCCESS 1
- #de£ne DB_CREATED 302
- #de£ne DB_NO_MEMORY 303
- #de£ne DB_INVALID_NAME 304
- #de£ne DB_INVALID_HANDLE 305
- #de£ne DB_NO_SLOT 306
- #de£ne DB_NO_MUTEX 307
- #de£ne DB_MEMSIZE_MISMATCH 308
- #de£ne DB_INVALID_PARAM 309
- #de£ne DB_FULL 310
- #de£ne DB_KEY_EXIST 311
- #de£ne DB_NO_KEY 312
- #de£ne DB_KEY_CREATED 313
- #de£ne DB_TRUNCATED 314
- #de£ne DB_TYPE_MISMATCH 315
- #de£ne DB_NO_MORE_SUBKEYS 316
- #de£ne DB_FILE_ERROR 317
- #de£ne DB_NO_ACCESS 318
- #de£ne DB_STRUCT_SIZE_MISMATCH 319
- #de£ne DB_OPEN_RECORD 320
- #de£ne DB_OUT_OF_RANGE 321
- #de£ne DB_INVALID_LINK 322
- #de£ne DB_CORRUPTED 323
- #de£ne DB_STRUCT_MISMATCH 324
- #de£ne DB_TIMEOUT 325
- #de£ne DB_VERSION_MISMATCH 326
- #de£ne SS_SUCCESS 1
- #de£ne SS_CREATED 402
- #de£ne SS_NO_MEMORY 403
- #de£ne SS_INVALID_NAME 404
- #de£ne SS_INVALID_HANDLE 405
- #de£ne SS_INVALID_ADDRESS 406
- #de£ne SS_FILE_ERROR 407
- #de£ne SS_NO_MUTEX 408
- #de£ne SS_NO_PROCESS 409
- #de£ne SS_NO_THREAD 410
- #de£ne SS_SOCKET_ERROR 411

- #de£ne SS_TIMEOUT 412
- #de£ne SS_SERVER_RECV 413
- #de£ne SS_CLIENT_RECV 414
- #de£ne SS_ABORT 415
- #de£ne SS_EXIT 416
- #de£ne SS_NO_TAPE 417
- #de£ne SS_DEV_BUSY 418
- #de£ne SS_IO_ERROR 419
- #de£ne SS_TAPE_ERROR 420
- #de£ne SS_NO_DRIVER 421
- #de£ne SS_END_OF_TAPE 422
- #de£ne SS_END_OF_FILE 423
- #de£ne SS_FILE_EXISTS 424
- #de£ne SS_NO_SPACE 425
- #de£ne SS_INVALID_FORMAT 426
- #de£ne SS_NO_ROOT 427
- #de£ne RPC_SUCCESS 1
- #de£ne RPC_ABORT SS_ABORT
- #de£ne RPC_NO_CONNECTION 502
- #de£ne RPC_NET_ERROR 503
- #de£ne RPC_TIMEOUT 504
- #de£ne RPC_EXCEED_BUFFER 505
- #de£ne RPC_NOT_REGISTERED 506
- #de£ne RPC_CONNCLOSED 507
- #de£ne RPC_INVALID_ID 508
- #de£ne RPC_SHUTDOWN 509
- #de£ne RPC_NO_MEMORY 510
- #de£ne RPC_DOUBLE_DEFINED 511
- #de£ne FE_SUCCESS 1
- #de£ne FE_ERR_ODB 602
- #de£ne FE_ERR_HW 603
- #de£ne FE_ERR_DISABLED 604
- #de£ne FE_ERR_DRIVER 605
- #de£ne HS_SUCCESS 1
- #de£ne HS_FILE_ERROR 702
- #de£ne HS_NO_MEMORY 703
- #de£ne HS_TRUNCATED 704
- #de£ne HS_WRONG_INDEX 705
- #de£ne HS_UNDEFINED_EVENT 706
- #de£ne HS_UNDEFINED_VAR 707
- #de£ne FTP_SUCCESS 1
- #de£ne FTP_NET_ERROR 802
- #de£ne 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<<0)
- #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

5.26 mrpc.c File Reference

5.26.1 Detailed Description

The Midas RPC £le

De£nition in £le mrpc.c.

Variables

- RPC_LIST rpc_list_library [ ]
- RPC_LIST rpc_list_system [ ]

5.27 mrpc.h File Reference

5.27.1 Detailed Description

The mrpc include £le

De£nition in £le mrpc.h.

De£nes

- #de£ne RPC_CM_SET_CLIENT_INFO 11000
- #de£ne RPC_CM_SET_WATCHDOG_PARAMS 11001
- #de£ne RPC_CM_CLEANUP 11002
- #de£ne RPC_CM_GET_WATCHDOG_INFO 11003
- #de£ne RPC_CM_MSG_LOG 11004
- #de£ne RPC_CM_EXECUTE 11005
- #de£ne RPC_CM_SYNCHRONIZE 11006
- #de£ne RPC_CM_ASCTIME 11007
- #de£ne RPC_CM_TIME 11008
- #de£ne RPC_CM_MSG 11009
- #de£ne RPC_CM_EXIST 11011
- #de£ne RPC_CM_MSG_RETRIEVE 11012
- #de£ne RPC_CM_MSG_LOG1 11013
- #de£ne RPC_BM_OPEN_BUFFER 11100
- #de£ne RPC_BM_CLOSE_BUFFER 11101
- #de£ne RPC_BM_CLOSE_ALL_BUFFERS 11102
- #de£ne RPC_BM_GET_BUFFER_INFO 11103
- #de£ne RPC_BM_GET_BUFFER_LEVEL 11104

- #de£ne RPC_BM_INIT_BUFFER_COUNTERS 11105
- #de£ne RPC_BM_SET_CACHE_SIZE 11106
- #de£ne RPC_BM_ADD_EVENT_REQUEST 11107
- #de£ne RPC_BM_REMOVE_EVENT_REQUEST 11108
- #de£ne RPC_BM_SEND_EVENT 11109
- #de£ne RPC_BM_FLUSH_CACHE 11110
- #de£ne RPC_BM_RECEIVE_EVENT 11111
- #de£ne RPC_BM_MARK_READ_WAITING 11112
- #de£ne RPC_BM_EMPTY_BUFFERS 11113
- #de£ne RPC_BM_SKIP_EVENT 11114
- #de£ne RPC_DB_OPEN_DATABASE 11200
- #de£ne RPC_DB_CLOSE_DATABASE 11201
- #de£ne RPC_DB_CLOSE_ALL_DATABASES 11202
- #de£ne RPC_DB_CREATE_KEY 11203
- #de£ne RPC_DB_CREATE_LINK 11204
- #de£ne RPC_DB_SET_VALUE 11205
- #de£ne RPC_DB_GET_VALUE 11206
- #de£ne RPC_DB_FIND_KEY 11207
- #de£ne RPC_DB_FIND_LINK 11208
- #de£ne RPC_DB_GET_PATH 11209
- #de£ne RPC_DB_DELETE_KEY 11210
- #de£ne RPC_DB_ENUM_KEY 11211
- #de£ne RPC_DB_GET_KEY 11212
- #de£ne RPC_DB_GET_DATA 11213
- #de£ne RPC_DB_SET_DATA 11214
- #de£ne RPC_DB_SET_DATA_INDEX 11215
- #de£ne RPC_DB_SET_MODE 11216
- #de£ne RPC_DB_GET_RECORD_SIZE 11219
- #de£ne RPC_DB_GET_RECORD 11220
- #de£ne RPC_DB_SET_RECORD 11221
- #de£ne RPC_DB_ADD_OPEN_RECORD 11222
- #de£ne RPC_DB_REMOVE_OPEN_RECORD 11223
- #de£ne RPC_DB_SAVE 11224
- #de£ne RPC_DB_LOAD 11225
- #de£ne RPC_DB_SET_CLIENT_NAME 11226
- #de£ne RPC_DB_RENAME_KEY 11227
- #de£ne RPC_DB_ENUM_LINK 11228
- #de£ne RPC_DB_REORDER_KEY 11229
- #de£ne RPC_DB_CREATE_RECORD 11230
- #de£ne RPC_DB_GET_DATA_INDEX 11231
- #de£ne RPC_DB_GET_KEY_TIME 11232
- #de£ne RPC_DB_GET_OPEN_RECORDS 11233
- #de£ne RPC_DB_FLUSH_DATABASE 11235

- #de£ne RPC_DB_SET_DATA_INDEX2 11236
- #de£ne RPC_DB_GET_KEY_INFO 11237
- #de£ne RPC_DB_GET_DATA1 11238
- #de£ne RPC_DB_SET_NUM_VALUES 11239
- #de£ne RPC_DB_CHECK_RECORD 11240
- #de£ne RPC_DB_GET_NEXT_LINK 11241
- #de£ne RPC_HS_SET_PATH 11300
- #de£ne RPC_HS_DEFINE_EVENT 11301
- #de£ne RPC_HS_WRITE_EVENT 11302
- #de£ne RPC_HS_COUNT_EVENTS 11303
- #de£ne RPC_HS_ENUM_EVENTS 11304
- #de£ne RPC_HS_COUNT_VARS 11305
- #de£ne RPC_HS_ENUM_VARS 11306
- #de£ne RPC_HS_READ 11307
- #de£ne RPC_HS_GET_VAR 11308
- #de£ne RPC_HS_GET_EVENT_ID 11309
- #de£ne RPC_EL_SUBMIT 11400
- #de£ne RPC_AL_CHECK 11500
- #de£ne RPC_AL_TRIGGER_ALARM 11501
- #de£ne RPC_RC_TRANSITION 12000
- #de£ne RPC_ANA_CLEAR_HISTOS 13000
- #de£ne RPC_LOG_REWIND 14000
- #de£ne RPC_TEST 15000
- #de£ne RPC_CNAF16 16000
- #de£ne RPC_CNAF24 16001
- #de£ne RPC_MANUAL_TRIG 17000
- #de£ne RPC_ID_WATCHDOG 99997
- #de£ne RPC_ID_SHUTDOWN 99998
- #de£ne RPC_ID_EXIT 99999

5.28 msystem.h File Reference

5.28.1 Detailed Description

The Midas System include £le

De£nition in £le msystem.h.

Data Structures

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

De£nes

- #de£ne DRI_16 (1<<0)
- #de£ne DRI_32 (1<<1)
- #de£ne DRI_64 (1<<2)
- #de£ne DRI_LITTLE_ENDIAN (1<<3)
- #de£ne DRI_BIG_ENDIAN (1<<4)
- #de£ne DRF_IEEE (1<<5)
- #de£ne DRF_G_FLOAT (1<<6)
- #de£ne DR_ASCII (1<<7)
- #de£ne WORD_SWAP(x)
- #de£ne DWORD_SWAP(x)
- #de£ne QWORD_SWAP(x)

5.29 mvmestd.h File Reference

5.29.1 De£ne Documentation

5.29.1.1 #de£ne EXPRT

De£nition at line 59 of £le mvmestd.h.

5.29.1.2 #de£ne MVME_A16D16 1

De£nition at line 77 of £le mvmestd.h.

5.29.1.3 #de£ne MVME_A16D32 2

De£nition at line 78 of £le mvmestd.h.

5.29.1.4 #de£ne MVME_A24D16 3

De£nition at line 79 of £le mvmestd.h.

5.29.1.5 #de£ne MVME_A24D32 4

De£nition at line 80 of £le mvmestd.h.

5.29.1.6 #de£ne MVME_A32D16 5

De£nition at line 81 of £le mvmestd.h.

5.29.1.7 #de£ne MVME_A32D32 6

De£nition at line 82 of £le mvmestd.h.

5.29.1.8 #de£ne MVME_AMOD_A16 MVME_AMOD_A16_SD

De£nition at line 125 of £le mvmestd.h.

5.29.1.9 #de£ne MVME_AMOD_A16_ND (0x29)

De£nition at line 123 of £le mvmestd.h.

5.29.1.10 #de£ne MVME_AMOD_A16_SD (0x2D)

De£nition at line 122 of £le mvmestd.h.

5.29.1.11 #de£ne MVME_AMOD_A24 MVME_AMOD_A24_SD

De£nition at line 119 of £le mvmestd.h.

5.29.1.12 #de£ne MVME_AMOD_A24_D64 MVME_AMOD_A24_SMBLT

De£nition at line 120 of £le mvmestd.h.

5.29.1.13 #de£ne MVME_AMOD_A24_NB (0x3B)

De£nition at line 113 of £le mvmestd.h.

5.29.1.14 #de£ne MVME_AMOD_A24_ND (0x39)

De£nition at line 115 of £le mvmestd.h.

5.29.1.15 #de£ne MVME_AMOD_A24_NMBLT (0x38)

De£nition at line 117 of £le mvmestd.h.

5.29.1.16 #de£ne MVME_AMOD_A24_NP (0x3A)

De£nition at line 114 of £le mvmestd.h.

5.29.1.17 #de£ne MVME_AMOD_A24_SB (0x3F)

De£nition at line 110 of £le mvmestd.h.

5.29.1.18 #de£ne MVME_AMOD_A24_SD (0x3D)

De£nition at line 112 of £le mvmestd.h.

5.29.1.19 #de£ne MVME_AMOD_A24_SMBLT (0x3C)

De£nition at line 116 of £le mvmestd.h.

5.29.1.20 #de£ne MVME_AMOD_A24_SP (0x3E)

De£nition at line 111 of £le mvmestd.h.

5.29.1.21 #de£ne MVME_AMOD_A32 MVME_AMOD_A32_SD

De£nition at line 107 of £le mvmestd.h.

5.29.1.22 #de£ne MVME_AMOD_A32_D64 MVME_AMOD_A32_SMBLT

De£nition at line 108 of £le mvmestd.h.

5.29.1.23 #de£ne MVME_AMOD_A32_NB (0x0B)

De£nition at line 101 of £le mvmestd.h.

5.29.1.24 #de£ne MVME_AMOD_A32_ND (0x09)

De£nition at line 103 of £le mvmestd.h.

5.29.1.25 #de£ne MVME_AMOD_A32_NMBLT (0x08)

De£nition at line 105 of £le mvmestd.h.

5.29.1.26 #de£ne MVME_AMOD_A32_NP (0x0A)

De£nition at line 102 of £le mvmestd.h.

5.29.1.27 #de£ne MVME_AMOD_A32_SB (0x0F)

De£nition at line 98 of £le mvmestd.h.

5.29.1.28 #de£ne MVME_AMOD_A32_SD (0x0D)

De£nition at line 100 of £le mvmestd.h.

5.29.1.29 #de£ne MVME_AMOD_A32_SMBLT (0x0C)

De£nition at line 104 of £le mvmestd.h.

5.29.1.30 #de£ne MVME_AMOD_A32_SP (0x0E)

De£nition at line 99 of £le mvmestd.h.

5.29.1.31 #de£ne MVME_IOCTL_AMOD_GET 3

De£nition at line 91 of £le mvmestd.h.

5.29.1.32 #de£ne MVME_IOCTL_AMOD_SET 2

De£nition at line 90 of £le mvmestd.h.

5.29.1.33 #de£ne MVME_IOCTL_CRATE_GET 1

De£nition at line 89 of £le mvmestd.h.

5.29.1.34 #de£ne MVME_IOCTL_CRATE_SET 0

De£nition at line 88 of £le mvmestd.h.

5.29.1.35 #de£ne MVME_IOCTL_DMA_GET 5

De£nition at line 93 of £le mvmestd.h.

5.29.1.36 #de£ne MVME_IOCTL_DMA_SET 4

De£nition at line 92 of £le mvmestd.h.

5.29.1.37 #de£ne MVME_IOCTL_FIFO_GET 7

De£nition at line 95 of £le mvmestd.h.

5.29.1.38 #de£ne MVME_IOCTL_FIFO_SET 6

De£nition at line 94 of £le mvmestd.h.

5.29.1.39 #de£ne MVME_LM 9

De£nition at line 85 of £le mvmestd.h.

5.29.1.40 #de£ne MVME_NO_CRATE 3

De£nition at line 66 of £le mvmestd.h.

5.29.1.41 #de£ne MVME_NO_INTERFACE 2

De£nition at line 65 of £le mvmestd.h.

5.29.1.42 #de£ne MVME_RAMD16 7

De£nition at line 83 of £le mvmestd.h.

5.29.1.43 #de£ne MVME_RAND32 8

De£nition at line 84 of £le mvmestd.h.

5.29.1.44 #de£ne MVME_SUCCESS 1

De£nition at line 64 of £le mvmestd.h.

5.29.1.45 #de£ne MVME_UNSUPPORTED 4

De£nition at line 67 of £le mvmestd.h.

5.29.1.46 #de£ne SUCCESS 1

De£nition at line 51 of £le mvmestd.h.

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

5.29.2 Typedef Documentation

5.29.2.1 typedef unsigned long int DWORD

De£nition at line 48 of £le mvmestd.h.

Referenced by bk_£nd(), bk_locate(), bk_swap(), bm_check_buffers(), cm_-cleanup(), cm_shutdown(), cm_transition(), cm_yield(), db_check_record(), db_-open_database(), db_sprintf(), eb_user(), eb_yfragment_add(), poll_event(), read_-scaler_event(), register_equipment(), rpc_send_event(), scaler_accum(), scheduler(), send_event(), source_scan(), ss_millitime(), ss_thread_create(), ss_thread_kill(), ss_-time(), update_odb(), ybk_close(), ybk_create(), ybk_£nd(), ybk_iterate(), ybk_list(), ybk_locate(), and ybk_size().

5.29.2.2 typedef unsigned long mvme_addr_t

De£nition at line 71 of £le mvmestd.h.

5.29.2.3 typedef unsigned long mvme_size_t

De£nition at line 72 of £le mvmestd.h.

5.29.2.4 typedef unsigned short int WORD

De£nition at line 43 of £le mvmestd.h.

Referenced by adc_calib(), bk_swap(), cm_cleanup(), cm_msg(), cm_msg1(), db_-open_database(), db_open_record(), db_sprintf(), read_trigger_event(), update_odb(), and ybk_close().

5.29.3 Function Documentation

5.29.3.1 int EXPRT mvme_exit ()

5.29.3.2 int EXPRT mvme_init ()

5.29.3.3 int EXPRT mvme_ioctl (int *req*, int ∗ *parm*)

5.29.3.4 int EXPRT mvme_mmap (void ** *ptr*, mvme_addr_t *vme_addr*, mvme_size_t *size*)

5.29.3.5 int EXPRT mvme_read (void * *dst*, mvme_addr_t *vme_addr*, mvme_size_t *size*)

5.29.3.6 int EXPRT mvme_unmap (void * *ptr*, mvme_size_t *size*)

5.29.3.7 int EXPRT mvme_write (mvme_addr_t *vme_addr*, void * *src*, mvme_size_t *n_bytes*)

5.30 newdocfeatures.dox File Reference

5.31 odb.c File Reference

5.31.1 Detailed Description

The Online Database £le

De£nition in £le 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_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 index, 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 index, 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 index, DWORD type)
- INT db_load (HNDLE hDB, HNDLE hKeyRoot, char *filename, BOOL b-Remote)
- 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_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 index, 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 socket)
- INT db_send_changed_records ()

5.32 odbstruct.dox File Reference

5.33 quickstart.dox File Reference

5.34 scaler.c File Reference

5.34.1 Function Documentation

5.34.1.1 INT scaler_accum (EVENT_HEADER ∗, void ∗)

De£nition at line 78 of £le scaler.c.

5.34.1.2 INT scaler_clear (INT *run_number*)

De£nition at line 63 of £le scaler.c.

5.34.1.3 INT scaler_eor (INT *run_number*)

De£nition at line 71 of £le scaler.c.

5.34.2 Variable Documentation

5.34.2.1 double scaler[32]

De£nition at line 59 of £le scaler.c.

Referenced by scaler_accum(), and scaler_clear().

5.34.2.2 ANA_MODULE scaler_accum_module

Initial value:

```
 {
   "Scaler accumulation",
   "Stefan Ritt",
   scaler_accum,
   scaler_clear,
   scaler_eor,
   NULL,
   NULL,
   NULL,
   0,
   NULL,
}
```

De£nition at line 44 of £le scaler.c.

5.35 system.c File Reference

5.35.1 Detailed Description

The Midas System £le

De£nition in £le system.c.

Functions

- 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)

5.36 utilities.dox File Reference

5.37 ybos.c File Reference

5.37.1 Detailed Description

The YBOS £le

De£nition in £le 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_£nd (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)

5.38 ybos.h File Reference

5.38.1 Detailed Description

The YBOS include £le

De£nition in £le ybos.h.

De£nes

- #de£ne YBOS_PHYREC_SIZE 8192
- #de£ne YBOS_BUFFER_SIZE 3∗(YBOS_PHYREC_SIZE<<2) + MAX_-EVENT_SIZE + 128
- #de£ne YB_BANKLIST_MAX 32
- #de£ne YB_STRING_BANKLIST_MAX YB_BANKLIST_MAX ∗ 4
- #de£ne YB_SUCCESS 1
- #de£ne YB_EVENT_NOT_SWAPPED 2
- #de£ne YB_DONE 2
- #de£ne YB_WRONG_BANK_TYPE -100
- #de£ne YB_BANK_NOT_FOUND -101

- #de£ne YB_SWAP_ERROR -102
- #de£ne YB_NOMORE_SLOT -103
- #de£ne YB_UNKNOWN_FORMAT -104
- #de£ne H_BLOCK_SIZE 0
- #de£ne H_BLOCK_NUM 1
- #de£ne H_HEAD_LEN 2
- #de£ne H_START 3
- #de£ne D_RECORD 1
- #de£ne D_HEADER 2
- #de£ne D_EVTLEN 3
- #de£ne YB_COMPLETE 1
- #de£ne YB_INCOMPLETE 2
- #de£ne YB_NO_RECOVER -1
- #de£ne YB_NO_RUN 0
- #de£ne YB_ADD_RUN 1
- #de£ne DSP_RAW 1
- #de£ne DSP_BANK 2
- #de£ne DSP_UNK 0
- #de£ne DSP_DEC 1
- #de£ne DSP_HEX 2
- #de£ne DSP_ASC 3
- #de£ne SWAP_D2WORD(_d2w)
- #de£ne EVID_TRINAT
- #de£ne YBOS_EVID_BANK(__a, __b, __c,__d,__e)
- #de£ne MIDAS_EVID_BANK(__a, __b, __c,__d,__e)
- #de£ne I2_BKTYPE 1
- #de£ne A1_BKTYPE 2
- #de£ne I4_BKTYPE 3
- #de£ne F4_BKTYPE 4
- #de£ne D8_BKTYPE 5
- #de£ne I1_BKTYPE 8
- #de£ne MAX_BKTYPE I1_BKTYPE+1

6 Midas Page Documentation

6.1 MIDAS Analyzer

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

- The Midas distribution contains 2 directories where prede£ned set of analyzer £les 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 speci£c 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 of¤ine following the information given further down. The analysis goal is to:
  1. Initialize the ODB with prede£ned (user speci£c) structure (experim.h).
  2. Allocate memory space for histogram de£nition (booking).
  3. Acquire data from the frontend (or data £le).
  4. Process the incoming data bank event-by-event through user speci£c code (module).
  5. Generate computed quantitied banks (in module).
  6. Fill (increment) prede£ned histogram with data available within the user code.
  7. Produce a result £le containing histogram results and computed data (if possible) for further replay through dedicated analysis tool (PAW, ROOT).

- The analyzer is structured with the following £les:
  - experim.h
    - ODB experiment include £le de£ning the ODB structure required by the analyzer.
  - analyzer.c: main user core code.
    - De£nes the incoming bank structures
    - De£nes 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.

– **Make£le**

  * Speci£c make£le 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 speci£c to the type of histogram package involved and requires the proper libraries to generate the executable. Each directory has its own **Make£le:**

  – **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 £le which should be present in the standard midas/<os>/lib directory. In order to have this £le (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 £le which should be present in the standatd midas/<os>/lib directory. This £le (hmana.o) doesn't require any speci£c library.
    * The analyzer build requires also at that stage to have access to some of the cernlib library £les (See HAVE_HBOOK).
  - **Analyzer Lite**
    * In the case private histogramming or simple analyzed data storage is requested, ROOT and HBOOK can be disabled by unde£ning both HAVE_ROOT and HAVE_HBOOK during the build.
    * This Lite version does't require any reference to the external histogramming package. Removal of speci£c de£nition 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 speci£c of the operation of the demo.

6.1.1 MultiStage Concept

In order to make data analysis more ¤exible, 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 speci£c 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 £rst 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 modi£ed 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 £rst 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:

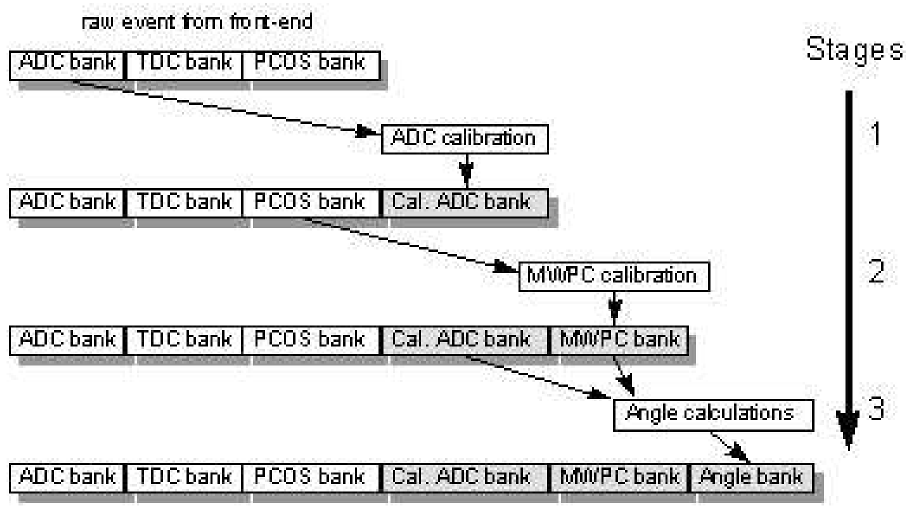


Figure 1: Three stage analyzer.

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 £le. 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 £tting 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" speci£ed in the analyzer.c under the analyzer_name. In case of concurrent analyzer, make sure that no con¤ict 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 £elds

```
[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 £le name.
  - **RWNT** : To be ignored for **ROOT**, N-Tuple Raw-wise data type.
  - **Histo Dump** : Enable the saving of the run results (see next £eld)
  - **Histo Dump Filename** : Online Result £le name
  - **Clear Histos** : Boolean ¤ag to enable the clearing of all histograms at the begining of each run (online or of¤ine).
  - **Last Histo Filename** : Temporary results £le for recovery procedure.
  - **Event to ODB** : Boolean ¤ag 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 £le. 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 £le. Used only during of¤ine, 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 ¤ag for booking N-Tuples at the initialization of the module. This ¤ag is speci£c to the **HBOOK** analyzer.
- **BOOK TTree** : Boolean ¤ag for booking TTree at the initialization of the module. This ¤ag is speci£c 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 de£nitions matched, the ODBEdit command **make** generates the £le experim.h which contains C structures for all the analyzer parameters. This £le is included in all analyzer source code £les and provides access to the parameters from within the module £le under the name <module name>_param.

- Module name: adc_calib_module (extern ANA_MODULE adc_calib_module from analyzer.c)
- Module £le 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_£le, -o output_£le for off-line use), the initialization phase will setup the following items:
  1. Setup the internal list of de£ned 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. De£ne 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 de£ned module.
5. Book the de£ned histograms of each module.
6. Book memory for N-Tuples or TTree.
7. Initialize the internal "hotlinks" to the de£ned 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 de£ned 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 de£ned 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 de£nes a section of 8 megabytes or 2 megawords of share memory for HBOOK/Midas data storage. This de£nition 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 £le 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 de£ned 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 ¤ag to indicate the run type for example, the analyzer can analyze calibration and data runs differently. The module declaration section in analyzer.c de£nes 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 de£ned in their source code £le. The enabled ¤ag for each module is copied to the ODB under /Analyzer/Module switches. By setting this ¤ag zero in the ODB, modules can be disabled temporarily. Next, all banks have to be de£ned. 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 £rst de£ned, 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 modi£ed 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
```

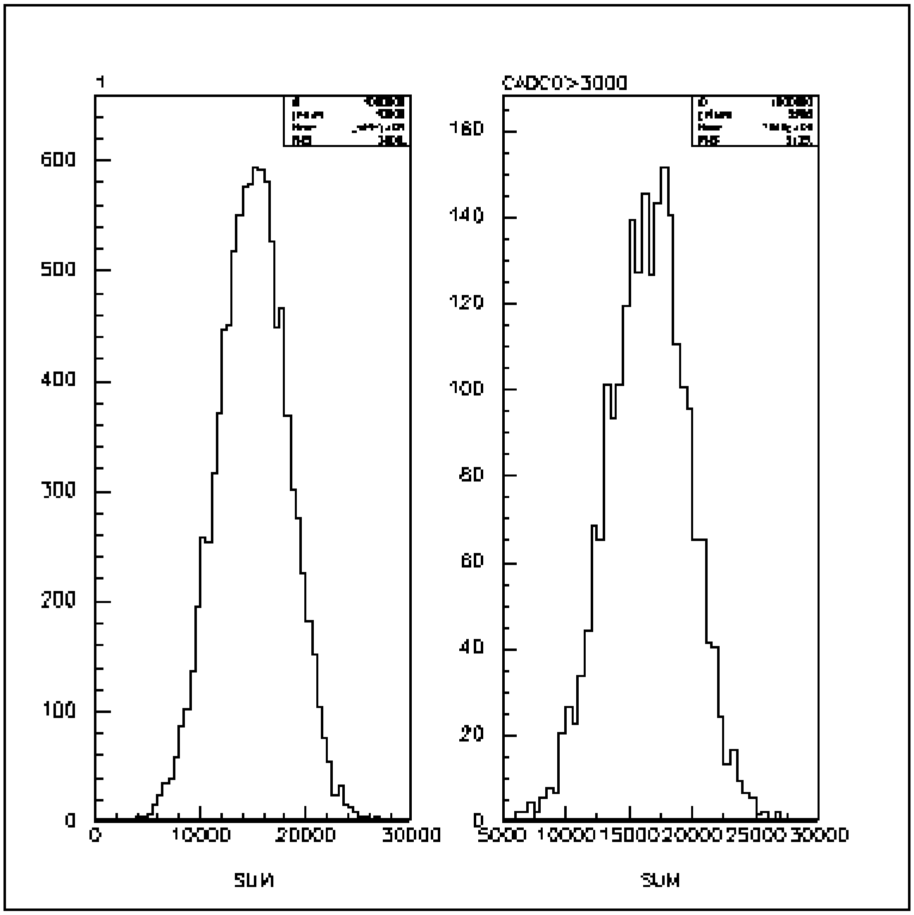


Figure 2: PAW output for online N-tuples.

While histograms contain the full statistics of a run, N-tuples are kept in a ring-buffer. The size of this buffer is de£ned 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 £rst 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 £lled. On the other hand care has to be taken in interpreting the data. If modi£cations in the hardware are made during a run, events which re¤ect the modi£cations 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 ¤ag and /Analyzer/Output/Histo Dump Filename determine if HBOOK histograms are written after a run. This £le 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 ¤ag 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 Of¤ine usage with PAW The analyzer can be used for off-line analysis without recompilation. It can read from MIDAS binary £les (∗.mid), analyze the data the same way as online, and the write the result to an output £le in MIDAS binary format, ASCII format or HBOOK RZ format. If written to a RZ £le, 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 ¤ags. Individual modules can be activated/deactivated via the /Analyzer/Module switches ¤ags.

The RZ £les can be analyzed and plotted with PAW. Following ¤ags are available when the analyzer is started off-line:

- -i [£lename1] [£lename2] ... Input £le name(s). Up to ten different £le names can be speci£ed in a -i statement. File names can contain the sequence "%05d" which is replaced with the current run number in conjunction with the -r ¤ag. Following £lename 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-¤y.
- -o [£lename] Output £le name. The £le names can contain the sequence "%05d" which is replaced with the current run number in conjunction with the -r ¤ag. Following £le formats can be generated: .mid (MIDAS binary), .asc (ASCII data), .rz (HBOOK RZ £le), .mid.gz (MIDAS binary gnu-zipped) and .asc.gz (ASCII data gnu-zipped). For HBOOK £les, CWNT are used by default. RWNT can be produced by specifying the -w ¤ag. Files are zipped on-the-¤y.
- -r [range] Range of run numbers to be analyzed like -r 120 125 to analyze runs 120 to 125 (inclusive). The -r ¤ag must be used with a "%05d" in the input £le 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 [£rst] [last] Analyze only events with serial numbers between £rst and last.

- -n [£rst] [last] [n] Analyze every n-th event from £rst to last.
- -c [£lename1] [£lename2] ... Load con£guration £le name(s) before analyzing a run. File names may contain a "%05d" to be replaced with the run number. If more than one £le is speci£ed, parameters from the £rst £le get superseded from the second £le 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 con£guration £le 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 speci£c value. Overrides any setting in con£guration £les. Parameter names are relative to the /Analyzer/Parameters directory. To set the key /Analyzer/Parameters/ADC summing/offset to a speci£c 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 £le. By default, column-wise N-tuples are used.
- -v Convert only input £le to output £le. Useful for format conversions. No data analysis is performed.
- -d Debug ¤ag 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 used by the consumers to distinguish between different events. The header is de£ned in the EVENT_HEADER structure in midas.h. It has following structure:

Event and bank headers with data block.

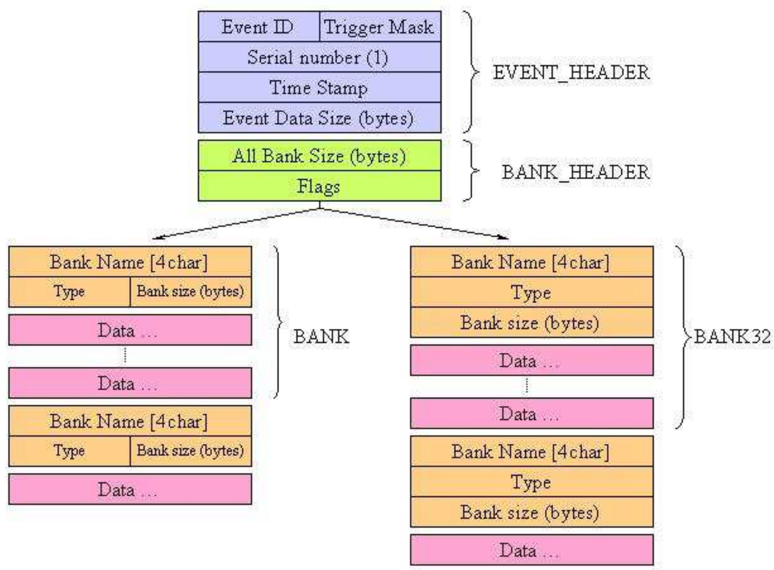


Figure 3: Event and bank headers with data block.

The event ID describes the type of event. Usually 1 is used for trigger 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 speci£c trigger 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 follow 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 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 identi£cation, a bank type that is one of the TID_xxx values de£ned 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 £les without any reformatting. For tapes, a £xed block size is used. The block size TAPE_BUFFER_SIZE is de£ned 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 identi£ed 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 a organized way. YBOS de£nes speci£c 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 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 £le 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.

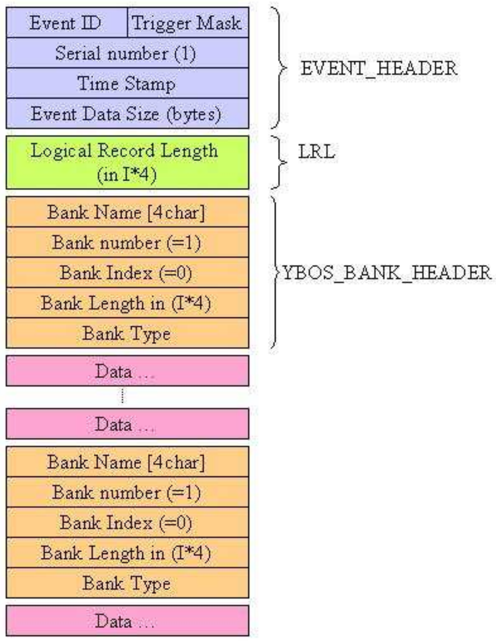


Figure 4: Ybos Event and bank headers with data block.

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

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

Ybos Physical record structure with data block.

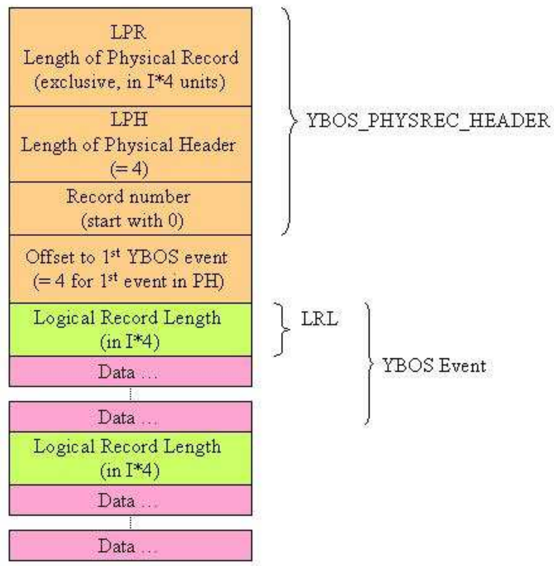


Figure 5: Ybos Physical record structure with data block..

The Offset is computed with the following rules:

- If the logical record £ts completely in the space of the physical record, the offset value in the physical record header will be 4.
- If the block contains £rst 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 £le is de£ned 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

Drivers included in the driver's directory of the MIDAS distribution support various hardware modules. The driver library is continuously extended to suit the needs of various experiments. For the slow control system. An example is 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) but includes directly the hardware calls. For some more explanation on the Slow control scheme, refer to Slow Control System Refer to the nulldev.c for a proper example.

Follows the class, device, bus and divers directory content under midas/drivers.

Class, Device, Bus and Divers Driver listing

Class	Device	Bus	Divers
generic.c	nulldev.c	null.c	caenv488.c
hv.c	epics_ca.c	vxVME.c	lrs1151.c
multi.c	lrs1454.c	camacrpc.c, camacnul.c	lrs1190.c
slowdev.c	dastemp.c	hyt1331.c	lrs2365.c
	lrs1440.c	kcs2926.c kcs2927.c	lrs2373.c
	bb_psi.c	jorway73a.c	ps7106.c
	lrs4032.c	wecc32.c	sis3803.c
	lcwp950.c	camaclx.c	sis3700.c
	mscbdev.c	dsp004.c	vmeio.c
	lrs2415.c	ces8210.c	
	nitronic.c	rs232.c	
	caen170a.c	tcpip.c	
	das1600.c	cc7700pci.c	
		esone.c	
		ces2117.c	
		lrs1821.c	
		str340.c	
		bt617.c	

Figure 6: Class, Device, Bus and Divers Driver listing

- CAMAC drivers
- VME drivers
- GPIB drivers
- Other drivers

6.3.1 CAMAC drivers

The CAMAC drivers can be used in different con£guration 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 £x 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 modi£ed 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 £rst load a module *camac-kcs292x*.o. This software is available but not part of the Midas distribution yet. Please contact `midas@triumf.ca` 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 £les.

- PCI/VME interface
  - [sis1100.c] PCI/VME with optical £ber 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 Scienti£c Discriminator.
- [ces8210.c] CES CAMAC crate controller.
- [vmeio.c] Triumf VMEIO General purpose I/O 24bits.

6.3.3 GPIB drivers

There is no speci£c 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@triumf.ca` for further information.

For GPIB Linux support please refer to `The Linux Lab Project`

6.3.4 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 de£nes its own set of CAMAC and VME 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 £le mcstd.h (Midas Camac Standard) provides the interface for the CAMAC access, the £le 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).

6.4.1 Midas CAMAC standard functions

Please refer to the £le below for function description. mcstd.h

6.4.2 ESONE CAMAC standard functions

Not all the functionality of ESONE standard have been fully tested

Please refer to the £le for function description.

esone.c

6.4.3 Midas VME standard functions

This interface is under revision for providing basic VME access through a independent software interface. Please refer to the £le below for speci£c function explanation.

mvmestd.h

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 unde£ned. 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.

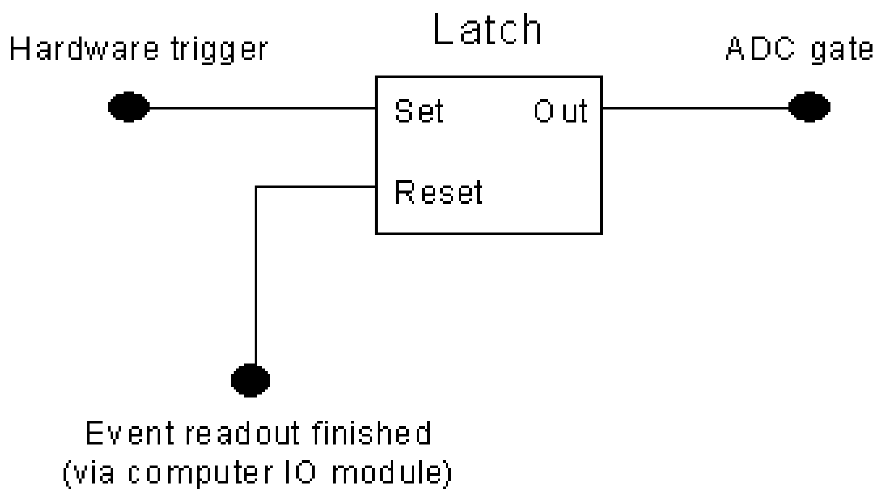


Figure 7: Latched trigger layout.

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

Improved Latched trigger layout.

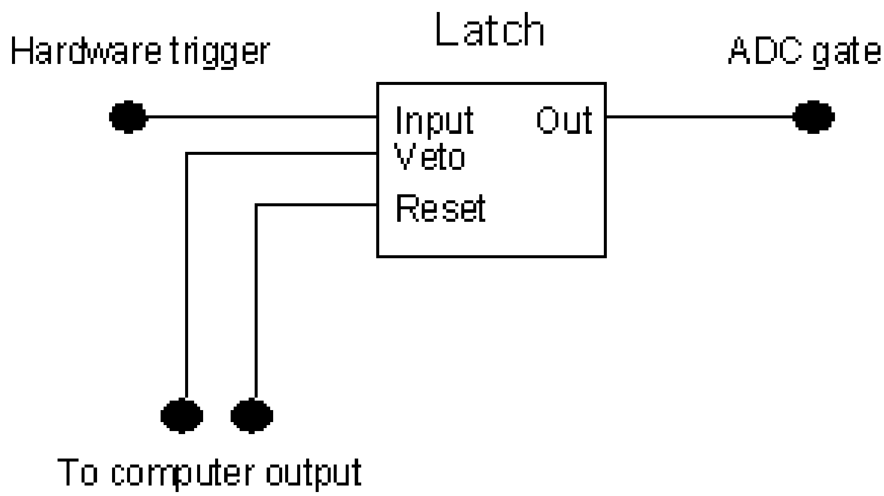


Figure 8: Improved Latched trigger layout.

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.

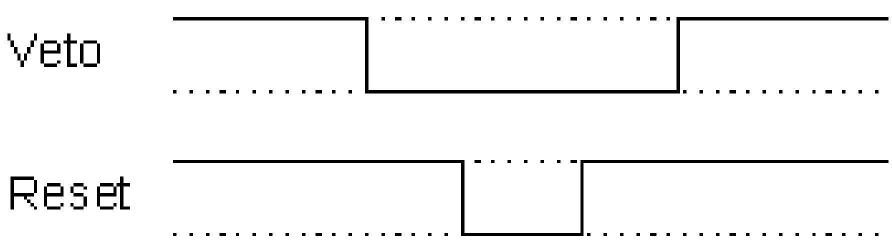


Figure 9: Veto Timing.

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 ¤ags. But the Make£le 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 ¤ags are avaiable at the application level:
  - HAVE_CAMAC , HAVE_ROOT , HAVE_HBOOK , HAVE_MYSQL , USE_EVENT_CHANNEL , DM_DUAL_THREAD , USE_INT
- By default the midas applications are built for use with dynamic library **libmidas.so**. If static build is required the whole package can be built using the option **static**.

  ```
  > make static
  ```

- The basic Midas package builds without external package library reference. But it does try to build an extra core analyzer application to be used in conjunction with ROOT if $ROOTSYS is found. This is required ONLY if the examples/experiment make£le 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 £x ¤ag established in the general Make£le. 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 Make£le as it requires a particular make£le and since several years now the VMS support has been dropped.

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

- Other OS supported are:
  - OS_WINNT : See £le make£le.nt.
  - OS_VXWORKS : See £le make£le.ppc_tri.

6.5.2 USERFLAGS

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

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

6.5.3 MIDAS_PREF_FLAGS

This ¤ag is for internal global Make£le preference. Included in the **OSFLAGS**.

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

6.5.4 HAVE_CAMAC

This ¤ag 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 unde£ned).

6.5.5 HAVE_ROOT

This ¤ag is used for the midas analyzer task in the case **ROOT** environment is required. An example of the make£le resides in **examples/experiment/Make£le**. This ¤ag 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 ¤ag is used for **examples/hbookexpt/Make£le** 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 make£le). This can always be overwritten during the make£le using the following command:

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

6.5.7 HAVE_MYSQL

This ¤ag is used for the mlogger task to building the application with *mySQL* support. The build requires to have access to the mysql include £les as well as the mysql library. Refers to for further information on that option.

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

6.5.8 SPECIFIC_OS_PRG

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

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

6.5.9 INCLUDE_FTPLIB

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

6.5.10 INCLUDE_ZLIB

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

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

6.5.11 YBOS_VERSION_3_3

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

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

6.5.12 DM_DUAL_THREAD

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

6.5.13 USE_EVENT_CHANNEL

To be used in conjunction with the DM_DUAL_THREAD.

6.5.14 USE_INT

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

6.5.15 Environment variables

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

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

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

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

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

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

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

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

6.6 Midas Code and Libraries

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

This section covers several aspect of the Midas system.

- 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 de£ned 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. ¤*oat* and *double* retain OS de£nition.

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

When de£ning a data type either in the frontend code for bank de£nition or in user code to de£ne ODB variables, Midas requires the use of its own data type declaration. The list below shows the main Type IDenti£cation to be used (refers to Midas De£ne 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 ¤oat format
- TID_DOUBLE eight bytes ¤oat 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 de£nes 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 De£ne)

- **I1_BKTYPE** Bank of Bytes
- **I2_BKTYPE** Bank of 2 bytes data
- **I4_BKTYPE** Bank of 4 bytes data
- **F4_BKTYPE** Bank of ¤oat 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 prede£ned macros (see Midas Code and Libraries) are available in order to extract from a pre-composed Midas event the internal header £elds i.e. Event ID, Trigger mask, Serial number, Time stamp. In this EVID bank we added the current run number which is retrieve by the frontend at the begin of a run.

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

INT read_cum_scaler_event(char *pevent, INT off)
{
  INT i;
  DWORD *pbkdat, *pbktop, *podbvar;

  ybk_init((DWORD *) pevent);

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

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

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

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

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

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

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

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

    // Swap event
    yb_any_event_swap(FORMAT_YBOS,plrl);

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

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

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

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

6.6.6 Midas Code and Libraries

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

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

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

- **al_xxx**(...) : Alarm system calls
- **bk_xxx**(...) : Midas bank manipulation calls
- **bm_xxx**(...) : Buffer management calls
- **cm_xxx**(...) : Common system calls
- **db_xxx**(...) : Database managment calls
- **el_xxx**(...) : Electronic Log book calls
- **hs_xxx**(...) : History manipulation calls
- **ss_xxx**(...) : System calls
- **ybk_xxx**(...) : YBOS bank manipulation

6.6.7 MIDAS Macros

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

- **Message Macros**. These Macros compact the 3 £rst 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 £rst. 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 £le resides?**

   - As soon as any midas application is started, a £le midas.log is produce. The location of this £le 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 £le **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 £le 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 modi£cation of a Midas £eld 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 £eld 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 £eld which needs to be set to **Read Only** £eld 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 £eld (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** £eld. 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 ¤exibility. Each component 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 brie¤y 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 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. The following link is a similar image with more information `Midas Structure`.

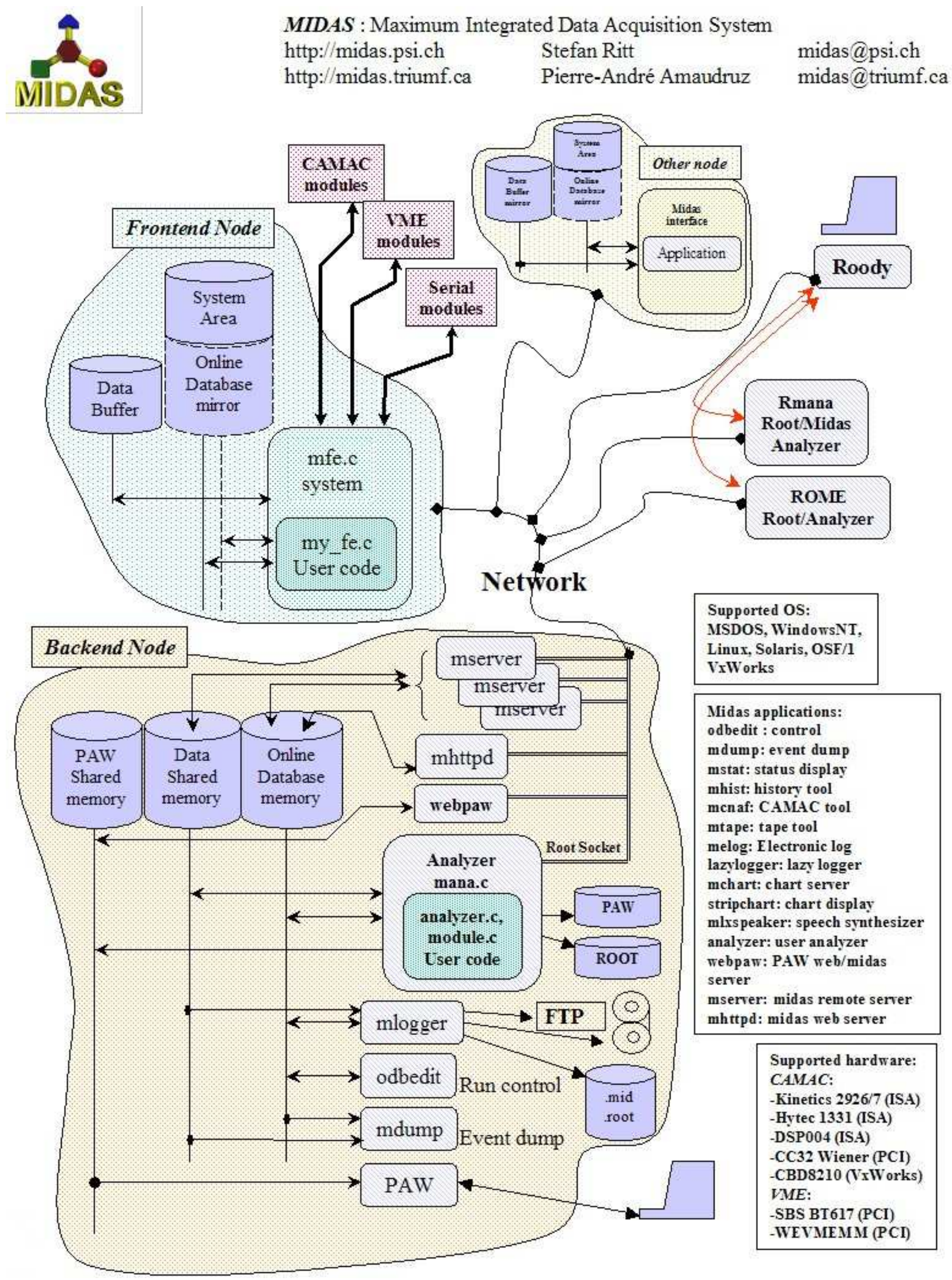


Figure 10: Components

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

- **Buffer Manager** Data ¤ow and messages passing mechanism.
- **Message System** Speci£c Midas messages ¤ow.
- **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 ¤ow 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 con£guration 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 share 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 in 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 £le system. The message system is based on the buffer manager scheme. A dedicated buffer is used to receive and distribute messages. Prede£ned message type contained in the Midas library covers most of the message requirement.

6.8.3 Online Database

In a distributed DAQ environment con£guration data is usually stored in several £les 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 £le 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 to a key can be of several type such as: byte, words, double words, ¤oat, 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 name, as 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 *frontend* 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 identi£ed event.

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

- Registers the given *Equipment(s)* list to the Midas system.
- Provides the mean of collecting the data from the hardware source de£ned in each equipment.
- Gathers these data in a known format (Fixed, Midas, Ybos) for each equipment.
- Sends these data to the buffer manager.
- Collects 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 con£guration.

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 £xed 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-de£ned 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 trigger 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 state is not paused and on all transitions" are possible.

Dedicated header and library £les 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 parallels logging channels with individual event selection criteria. Data can currently be written in £ve 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 event based on Event Identi£er.
- 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 £le 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 build for private analysis (no external analyzer hooks) or speci£c analysis packaged 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 are 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, 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 £le). 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 £les, analyze the events, add calculated data for each event and produce a HBOOK RZ output £le 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 ¤exibility of the ODB is available to the user's £ngertips.

Three "Run State de£nes 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 noti£cation of such message will be able to perform its task within the overall run control of the experiment.

In Order to provide more ¤exibility 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

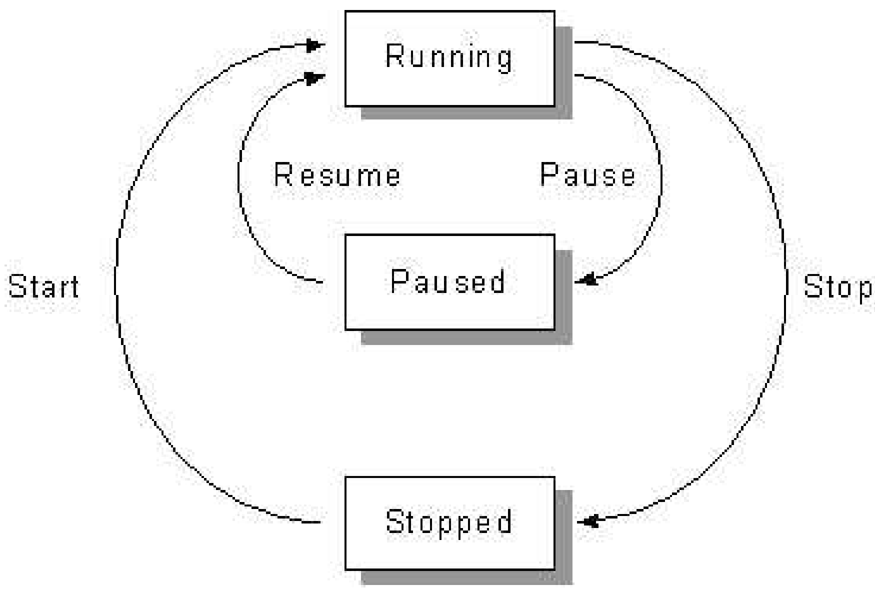


Figure 11: Transitions

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 modi£ed value gets automatically propagated to the slow control system, which in turn uses speci£c device driver to control the particular hardware. Measured values from the hardware are periodically send back to the ODB to re¤ect 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 speci£c 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. For more information see Slow Control.

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 de£ned channels, the Midas logger can store slow control data and/or periodic events on disk £le. 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 ¤exibility of the history system such as dynamic modi£cation of the event structure without restarting the Midas logger. At any given time, data-over-time relation can be queried from the disk £le through a Midas utility mhist task or displayed through the mhttpd task.

The history data extraction from the disk £le is done using low level £le 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 de£ned in the Online Database (ODB). Currently the internal alarms supports the following mechanism:

- ODB value over £xed threshold At regular time interval, a pre-de£ned ODB value will be compared to a £xed value.
- Midas client control During Run state transition, pre-de£ned Midas client name will be checked if currently present.
- General C-code alarm setting Alarm C function permitting to issue user de£ned 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 provide to 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 modi£cation 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 modi£ed.

```
> 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 ¤owchart 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.

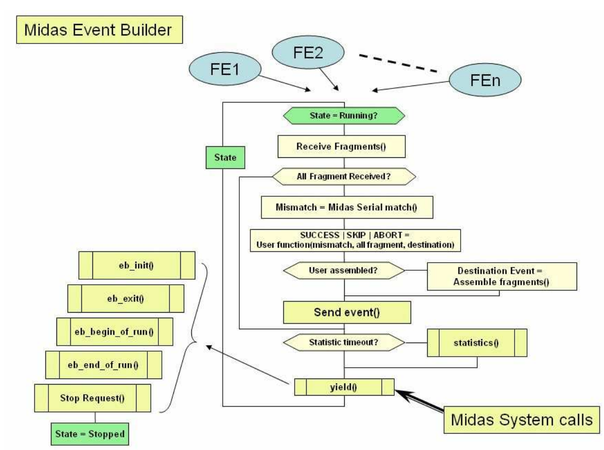


Figure 12: Event Builder Flowchart.

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 speci£c 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 de£ned experiment. – If no experiment de£ned 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 : Noti£cation mechanism
- Alarm System : Alarm scheme
- Slow Control System : Speci£c Slow Control mechanism
- Electronic Logbook : Essential utility
- Log £le : Message, error, report

6.10.1 Run Transition Sequence

The run transition sequence has been modi£ed 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 de£nes 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 noti£cation for run transition. This noti£cation 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 latest 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 re¤ects 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 to 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 £le, others of a C £le plus a header £le. The £le 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 between different platform as well as different CAMAC hardware interface without the need of modi£cation 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 remove.
  - frontend_name This value can be modi£ed to re¤ect the purpose of the code.
  - frontend_call_loop() Enables the function frontend_loop() to be run after every equipment loop.
  - display_period de£ned 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 the 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 £rst group of prototype(7) declare the pre-de£ned system functions should be present. The second group de£nes the user functions associated to the declared equipments. All the £elds 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 it-self 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** 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 de£nition]** Since the introduction of **ROOT** , the frontend requires to have the de£nition 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 de£nition]** 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 alos 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 £le.

```
    // 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 de£nition is **EQ_POLLED** as an acquisition type, the **poll_event()** will be call 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 station and is independent of the equipment, the LAM test can be simpli£ed (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 source is 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 station 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 include. 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 £le.

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

  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 £le.

```
// 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 de£nition. Here is the structure listing 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 de£ned 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 an unique event ID and to 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 modi£ed dynamically by the readout routine to de£ne 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 speci£ed 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 event coming from the frontend are called fragment). In this example both events are placed in the same buffer called "SYSTEM" (default).

  - **[Remark]** If this £eld 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 con£guration is to allow that particular equipment to be mirrored in the ODB if the RO_ODB is turned on.

- **[EQ_xxx]** The £eld specify the type of equipment. It can be composed of several bitwise ¤ags. The following EQ_POLLED, EQ_INTERRUPT and EQ_SLOW ¤ags cannot be Ored togheter. The possible options are:

  - [EQ_POLLED] In this mode, the name of the routine performing the trigger check function is defaulted to poll_event(). As polling consists on checking a variable for a true condition, if the loop would be in£nite, the frontend would not be able respond to any network commands. Therefore the loop count is determined when the frontend starts so that it returns after a given time-out when no event is available. This time-out is usually in the order of 500 milliseconds. This ¤ag 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 con£guration and control of the interrupt source. This is provided by an interrupt con£guration routine interrupt_con£gure() 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 con£guration 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" £eld in the equipment declaration is in this case used for periodicity.
- [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 ¤ag 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 ¤ag enables large event (beyond Midas con£guration limit) to be handled by the system. This ¤ag requires to have a valid **max_event_size_frag** variable de£ned 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 experiment 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 will be required to run, a declared function will be call 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 a 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 ¤ag identify the equipment as a **fragment event** and should be ored with the EQ_POLLED in order to be identifyed 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 de£nes which CAMAC slot is allowed to trigger the call to the readout routine. (See read_trigger_event()).

- **["MIDAS"]** This line speci£es 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 de£nition.

- [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 ¤ags 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 ¤ag 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 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 £eld 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. Modi£cation of "Enabled" ¤ag, RO_xxx ¤ags, "period" and "event limit" from the ODB is immediately re¤ected 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-de£ned through the interrupt_-con£guration has occurred.

– [Remark 1] The £rst argument of the readout function provide the pointer to the newly constructed event and point to the £rst valid location for storing the data.

– [Remark 2] The content of the memory location pointed by **pevent** prior 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 above example, the Midas Event Header will contains the same Event ID as well 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 £xed and maps to a C structure that is £lled by the readout routine. Since the standard MIDAS analyzer cannot work with this format, it is only recommended for experiment, which use its own analyzer and want to avoid the overhead of a bank structure. For £xed events, the structure has to be de£ned 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 de£ne a £xed 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 de£ned 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 £lled 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 4 £elds 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. Beside the variable data length support of the bank structure, onother main advantage is the possibility for the analyzer to add more (calculated) banks during the analysis process to the event in process. After the £rst 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() Locate a bank within anevent by its name.
- bk_iterate() Return bank and data pointers to each bank in the event.
- bk_list() Construct a string of all the bank name 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 £lled with speci£c 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 £lter (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 different 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 satis£ed. 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 postpone 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 satis£ed. 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 ¤ag £rst 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 satis£ed (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 a 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 £rst 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 see the following example:

- De£ne 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 decryption of the data section is left to the user. If the number of words per sub-event is £xed (NWORD), the sub-event extraction is simple. In the case of variable sub-event length, it is necessary to tag the £rst 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 guaranty 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 with 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 modi£ed, the frontend receives a noti£cation 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 signi£cant change. This design has a possible inconvenience due to fact that ODB is the key element of that control. Any failure or corruption of the database can results 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 between 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 use the same framework as the normal frontend and behave 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

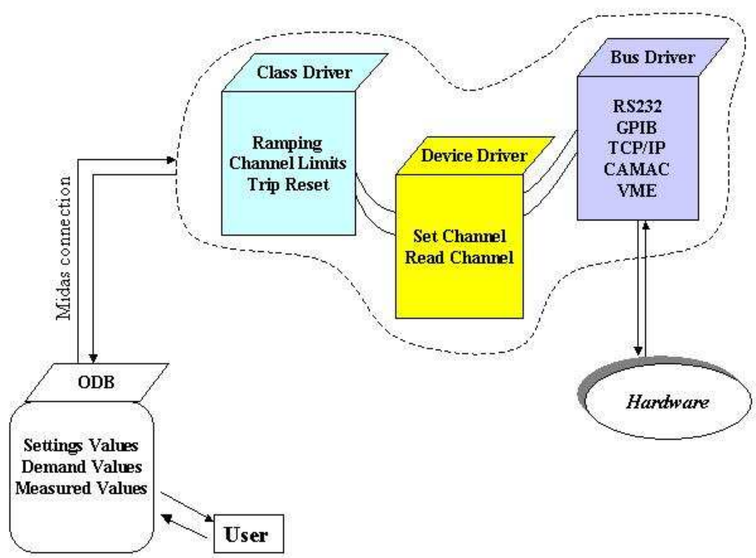


Figure 13: Class driver, Device and Bus driver in the slow control system

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. he 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 re¤ected 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 de£ned in the slow control frontend. Several equipment with different class drivers can be de£ned in a single frontend.

```
Key name                        Type    #Val  Size  Last Opn Mode Value
---------------------------------------------------------------------------
Epics                           DIR
    Settings                    DIR
        Channels                DIR
            Epics               INT     1     4     25h  0   RWD  3
        Devices                 DIR
            Epics               DIR
                Channel name    STRING  10    32    25h  0   RWD
                                        [0]             GPS:VAR1
                                        [1]             GPS:VAR2
                                        [2]             GPS:VAR3
        Names                   STRING  10    32    17h  1   RWD
                                        [0]             Current
                                        [1]             Voltage
                                        [2]             Watchdog
        Update Threshold 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 de£nition of the options can be found in the ODB data base under ODB /Elog Tree.

6.10.9 Log £le

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

1. [without ODB /Logger Tree] In this case the location is de£ned by either the MIDAS_DIR environment (see Environment variables) or the de£nition of the experiment in the **exptab** £le (see Experiment_De£nition). In both case the log £le will be in the experiment speci£c directory.

2. [with /Logger Tree] The **midas.log** will be sitting into the de£ned directory speci£ed by **Data Dir** .

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

The MIDAS Macros de£nition 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 £nding 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 £eld.

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 **Con£g-** button). This mean the information at any given time may re¤ect the experiment state of up to n second in the paste, where n is the timer setting of the refresh parameter. Its basic functionality are:

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

Each section is further described below:

- Start page - Run control page
- ODB page - Online Database manipulation (equivalent to ODBedit)
- Equipment page (Frontend info)
- CNAF page (CAMAC access page)
- Message page (Message Log)
- Elog page (Electronic Log)
- Program page (Program control)
- History page (History display)
- Alarm page (Alarm control)
- Custom page (User de£ned 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 £rst 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 £eld (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 (£le) 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 speci£c 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-de£ned variables. The history setting has to be done through ODB under the **/History** (see History system , History page).
- [Con£g 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 de£ne a new tree in ODB named **Script** than any subtree 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 de£nes a script names **doit** with 2 **Arguments** (run# device) which will be invoked when the button <doit> is pressed.

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

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

- **Example** : The **Example** will create a shortcut to the de£ned location in the ODB.

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

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

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

- [General info] Current run number, state, General Enable ¤ag for Alarm, Auto restart ¤ag 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 host-name on which their running.

Midas Web server

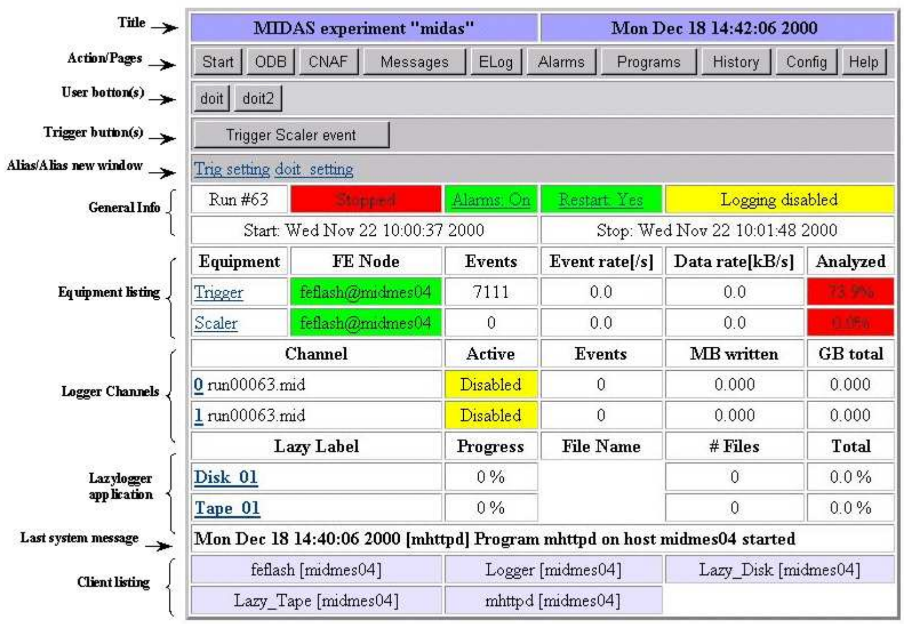
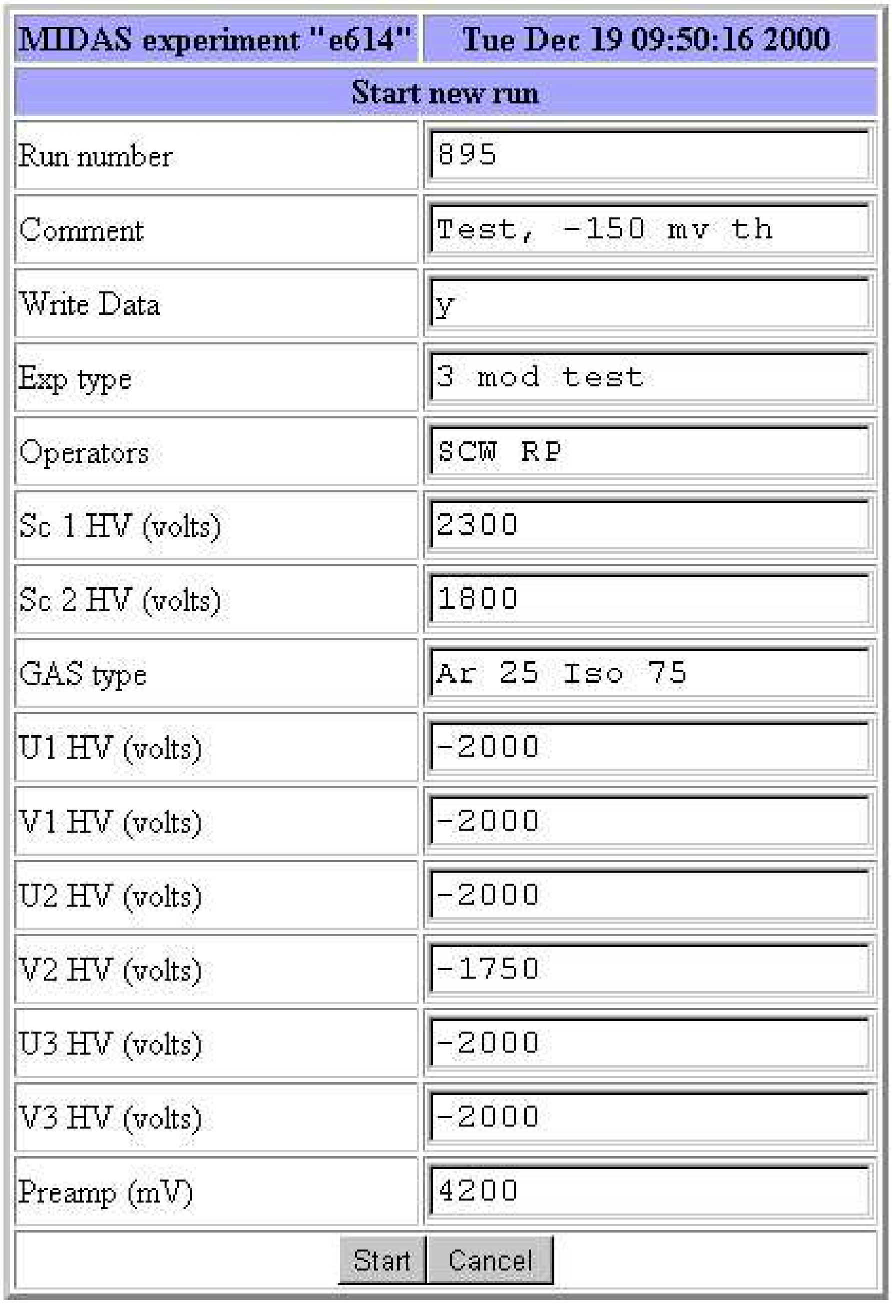


Figure 14: Midas Web server

6.12.1 Start page

Once the **Start** button is pressed, you will be prompt for experiment speci£c 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 de£ned the ODB tree **/Experiment/Edit on Start** all the parameters sitting in this directory will be displayed for possible modi£cation. 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 de£ned under "/Experiment/Edit on Start"



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

Figure 15: Start run request page.

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

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

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

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

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

Figure 16: Start run request page.

6.12.2 ODB page

The ODB page shows the ODB root tree at £rst. Clicking on the hyperlink will walk you to the requested ODB £eld. 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 £rst window will request the web password.

ODB page access.

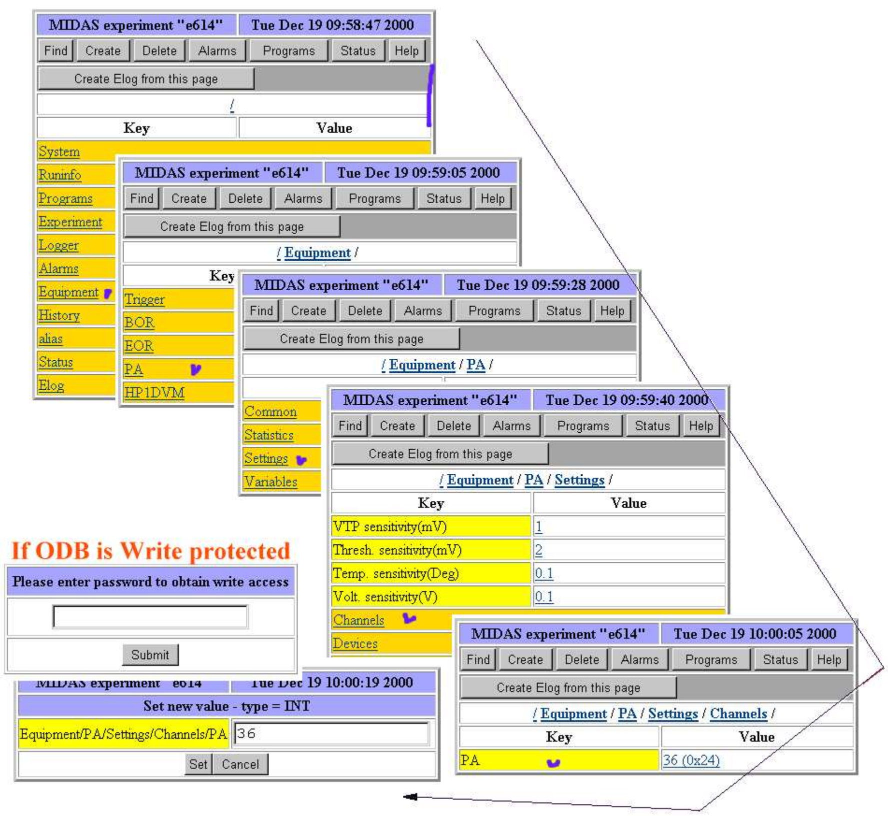
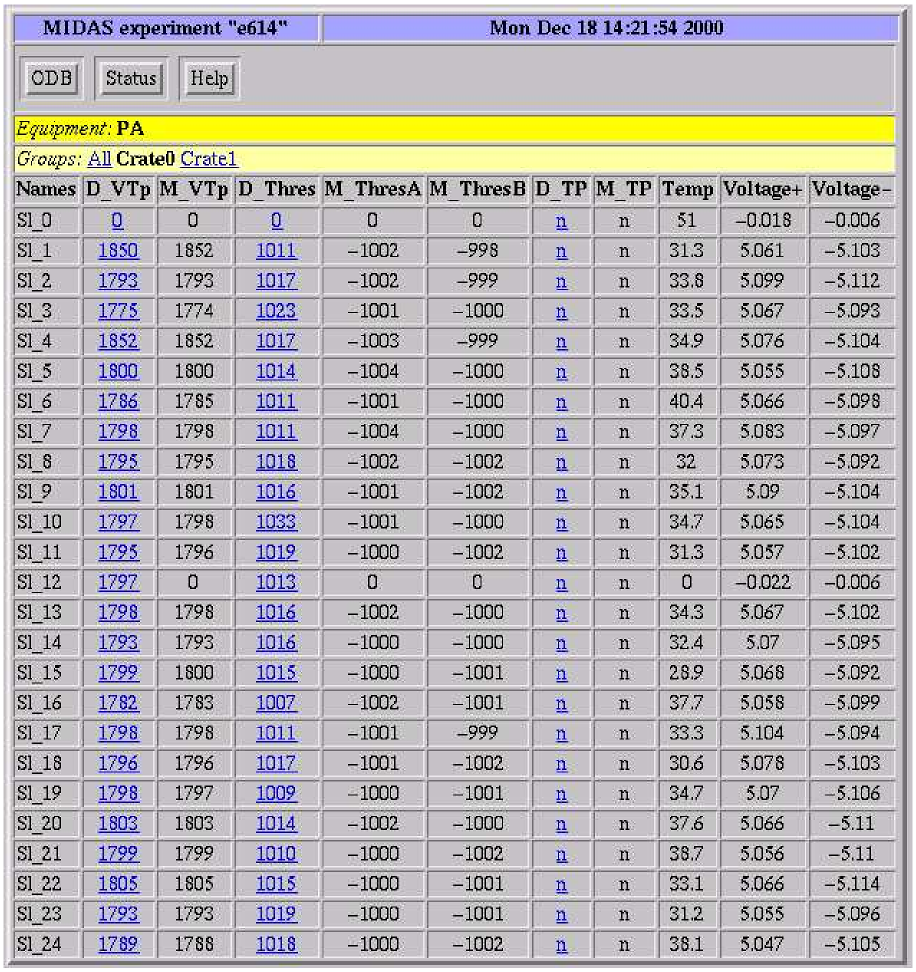


Figure 17: ODB page access.

6.12.3 Equipment page

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

Slow control page.



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

ODB Status Help

Equipment: **PA**

Groups: All **Crate0** Crate1

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

Figure 18: Slow control page.

6.12.4 CNAF page

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

CAMAC command pages.

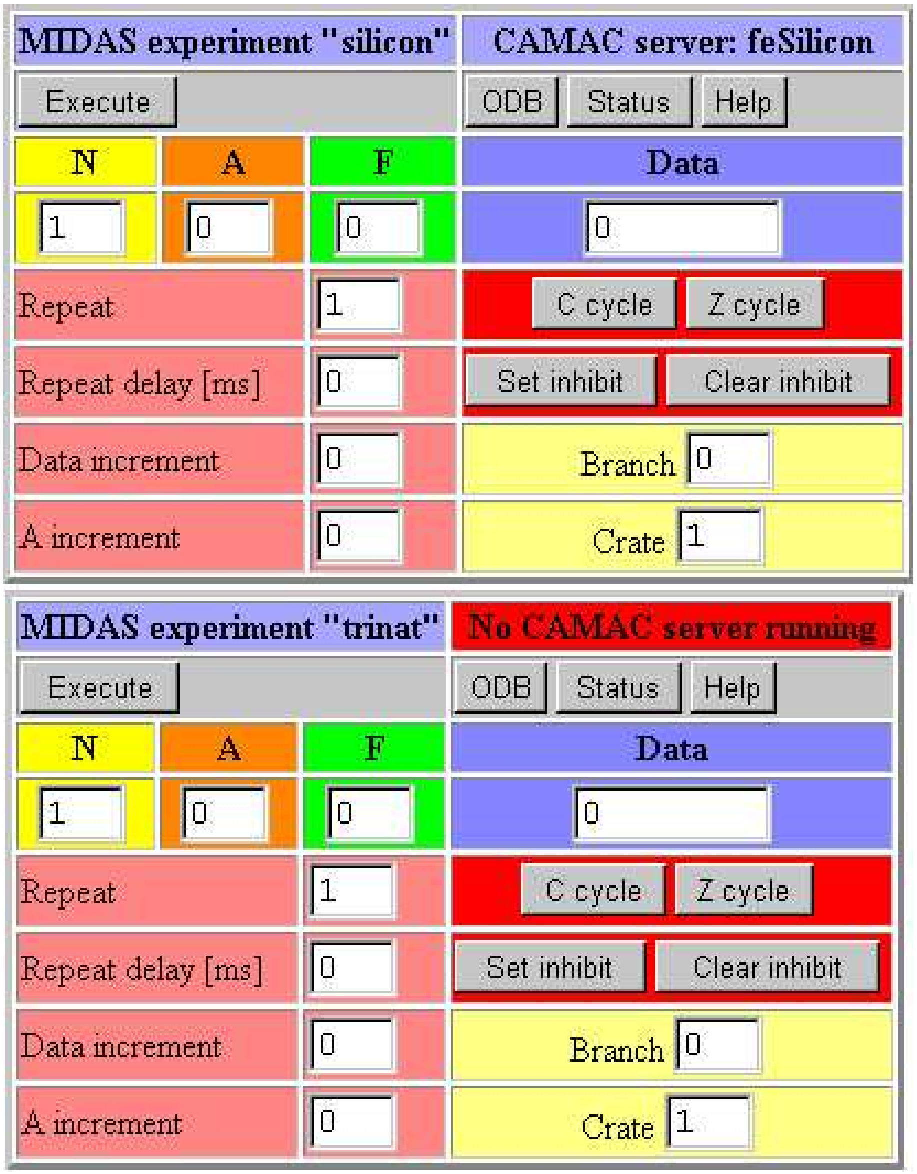


Figure 19: CAMAC command pages.

6.12.5 Message page

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

Message page.

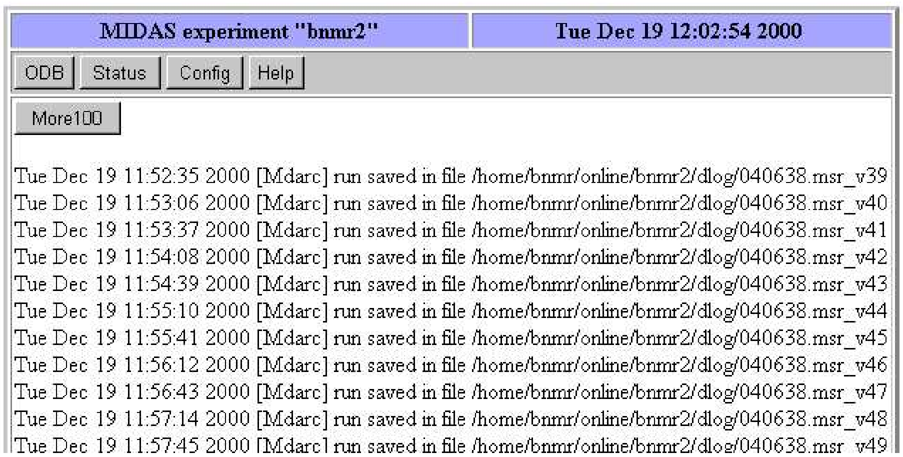


Figure 20: Message page.

6.12.6 Elog page

The Electronic Log page show the most recent Log message recorded in the system. The top buttons allows you to either Create/Edit/Reply/Query/Show a message.

From version 1.9.0, this page contains extra parameters for time selection and Email forwarding. Please refer to the ODB structure for further information.

main Elog page.

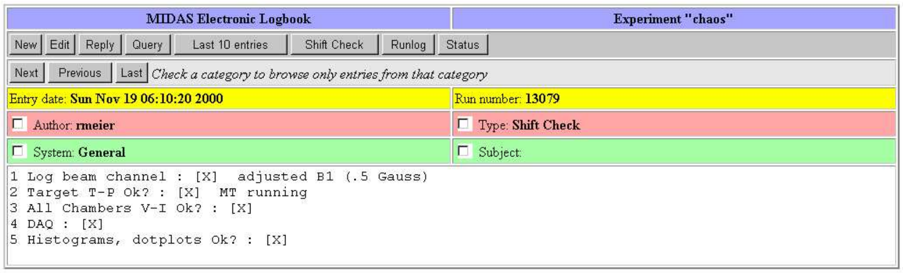


Figure 21: main Elog page.

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

HTML Elog message.

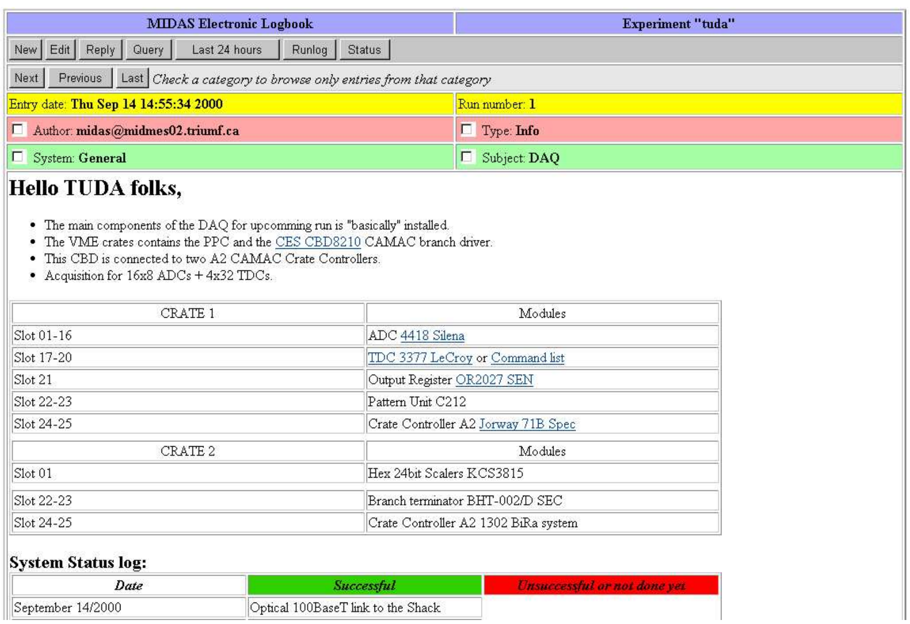


Figure 22: HTML Elog message.

The **runlog** button display the content of the £le **runlog.txt** which is expected to be in the data directory speci£ed 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 **Example** /experiment/analyzer.c which at each end-of-run (EOR) will write to the runlog.txt some statistical informations.

Elog page, Runlog display.

MIDAS File Display | Experiment "ltno"

ELog Status

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 23: Elog page, Runlog display.

When composing a new entry into the Elog, several £elds 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 de£nable through the odb under the tree **/Elog/Types**, **/Elog/Systems**. The number of category is £xed to 20 maximum but any remaining £eld can be left empty.

Elog page, New Elog entry form.

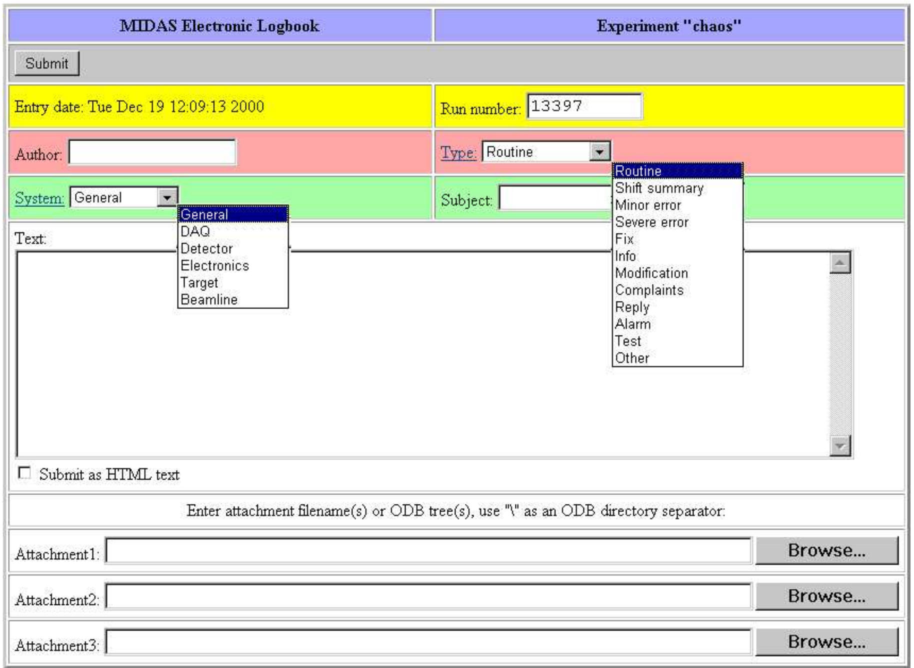


Figure 24: Elog page, New Elog entry form.

6.12.7 Program page

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

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

Program page.

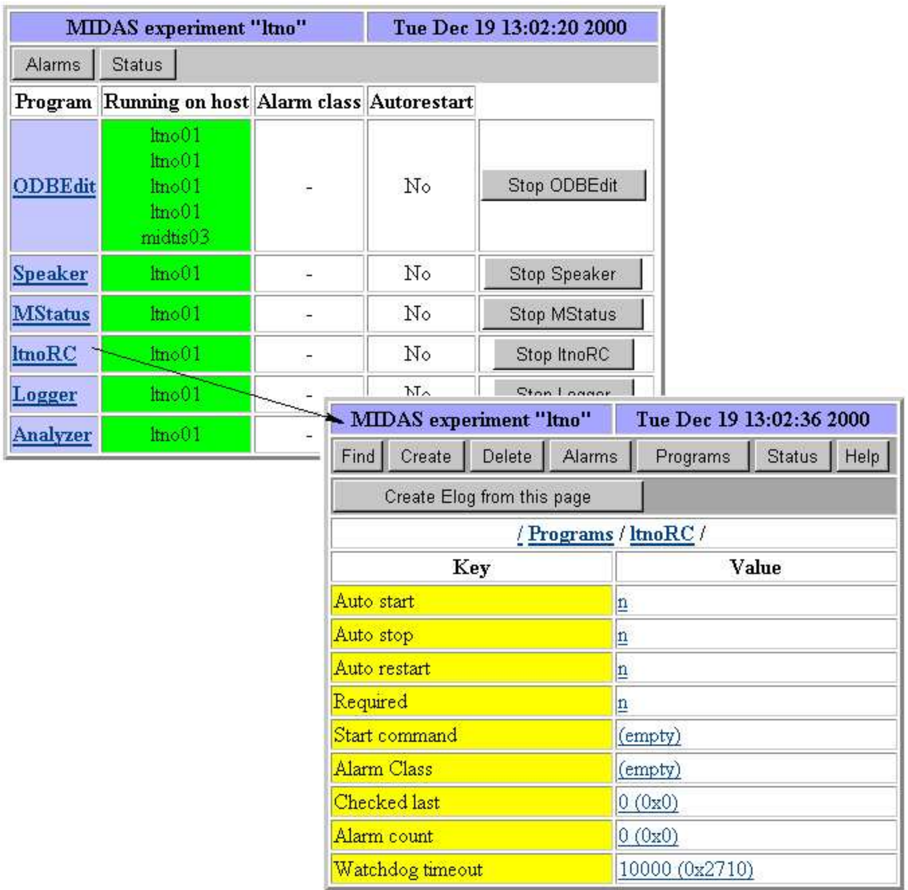


Figure 25: Program page.

6.12.8 History page

This page re¤ects the History system settings (CVS r1.271). It lists on the top of the page the possible group names containing a list of panels de£ned in the ODB. Next a serie of buttons de£nes the time scale of the graph with prede£ned time window, "<<","<" "+" "-" ">" ">>" buttons permit the shifting of the graph in the time direction. Other buttons will allow graph resizing, Elog attachment creation, con£guration 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 con£guration 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 26: History page.

History channel selection Page.

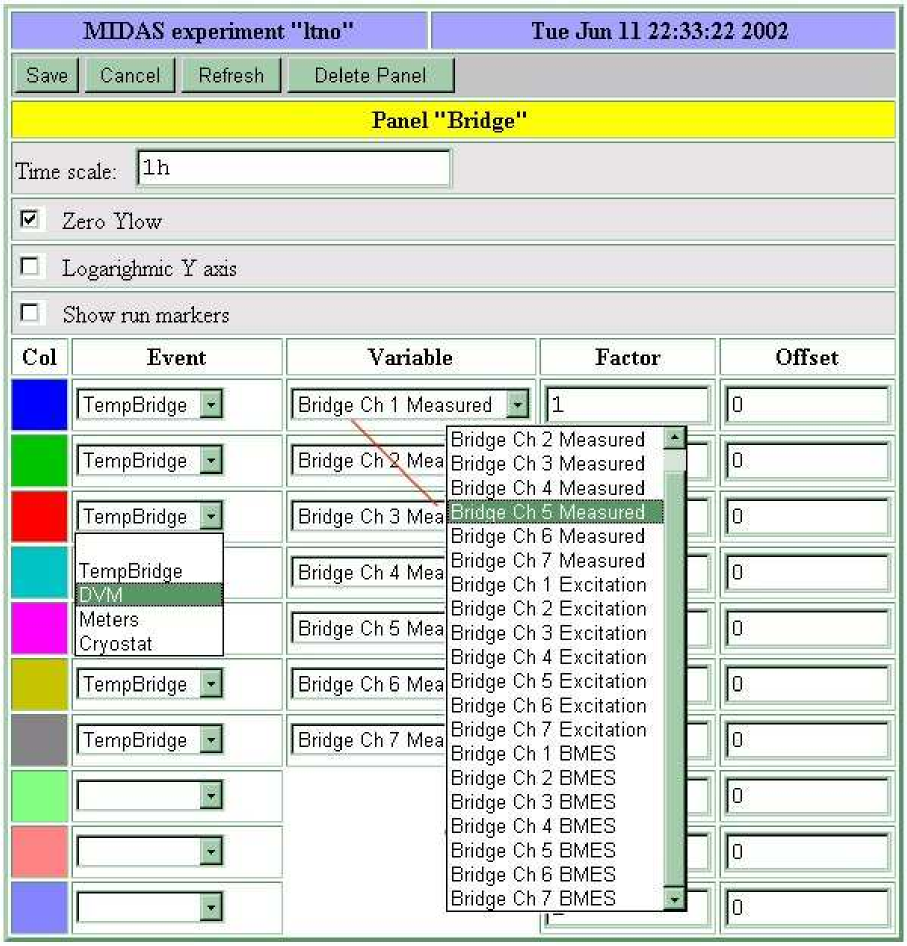


Figure 27: History channel selection Page.

6.12.9 Alarm page

This page re¤ects 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 de£ned task.
- [Internal alarms] Trigger on internal (program) alarm setting through the use of the *al_*...() functions.

- [Periodic alarms] Triggered by timeout condition de£ned in the alarm setting.

6.12.10 Custom page

The Custom page is available from the Version 1.8.3.

This page re¤ects the html content of a given ODB key under the **/Custom/** key. If keys are de£ned in the ODB under the **/Custom/** the name of the key will appear in the main status page as the **Alias** keys. By clicking on the Custom page name, the content of the **/Custom/<page>** is interpreted as html content.

Custom web page with history graph.

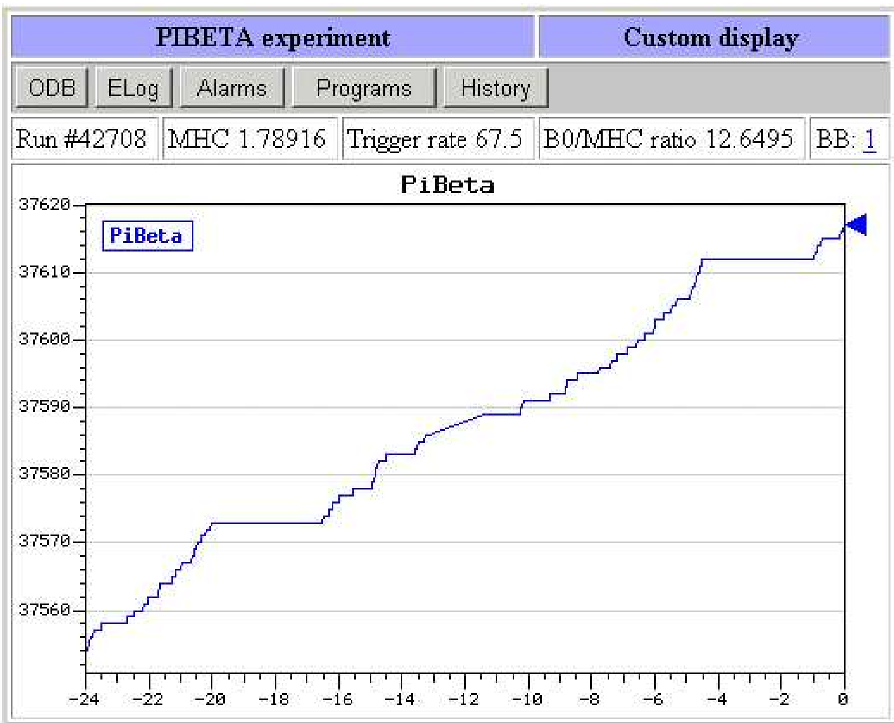


Figure 28: Custom web page with history graph.

The access to the ODB £eld is then possible using speci£c HTML tags:

- **<odb src="odb £eld">** Display ODB £eld.

- <**odb src="odb £eld" edit=1**> Display and Editable ODB £eld.
- <form method="GET" action="http://hostname.domain:port/CS/<Custom_-page_key>"> De£ne method for key access.
- <meta http-equiv="Refresh" content="60"> Standard page refresh in second.
- <input type=submit name=cmd value=<Midas_page>> De£ne 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 £eld.

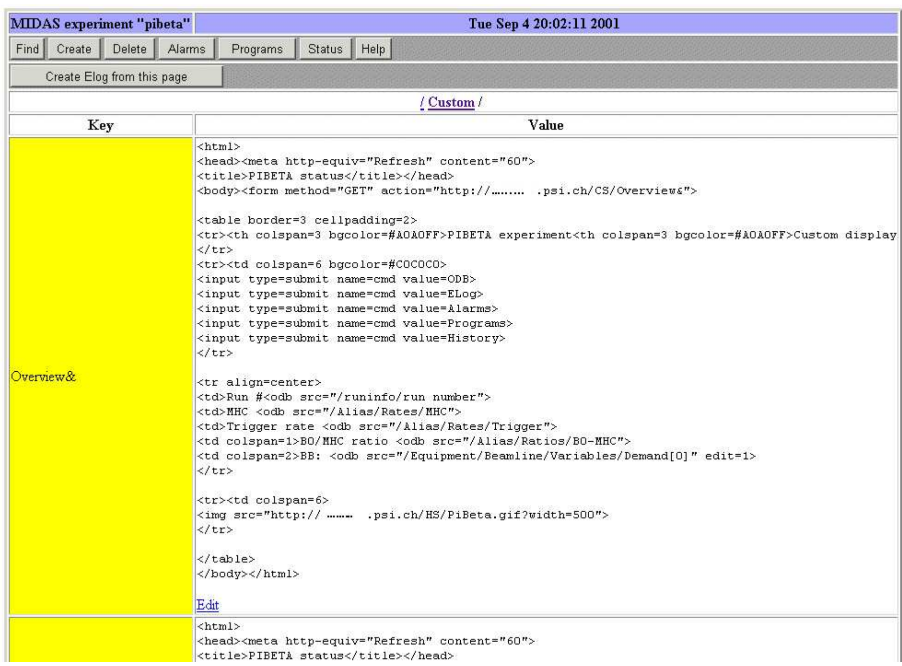


Figure 29: ODB /Custom/ html £eld.

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

- Create an initial html £le using your prefered HTML editor.
- Insert the ODB HTML tags at your wish.
- Invoke ODBedit, create the Custom directory, import the html £le.

- **Example** of loading the £le 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 £le 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 £le 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>

<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">
<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">
<br>Run#: <odb src="/runinfo/run number">
<br>Run#: <odb src="/runinfo/run number">
<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">
<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">
<br>Run#: <odb src="/runinfo/run number">
<br>Run#: <odb src="/runinfo/run number">
<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">
<br>Run#: <odb src="/runinfo/run number">
<br>Run#: <odb src="/runinfo/run number"></td>

<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"></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>
```

web page produced by mcustom.html.

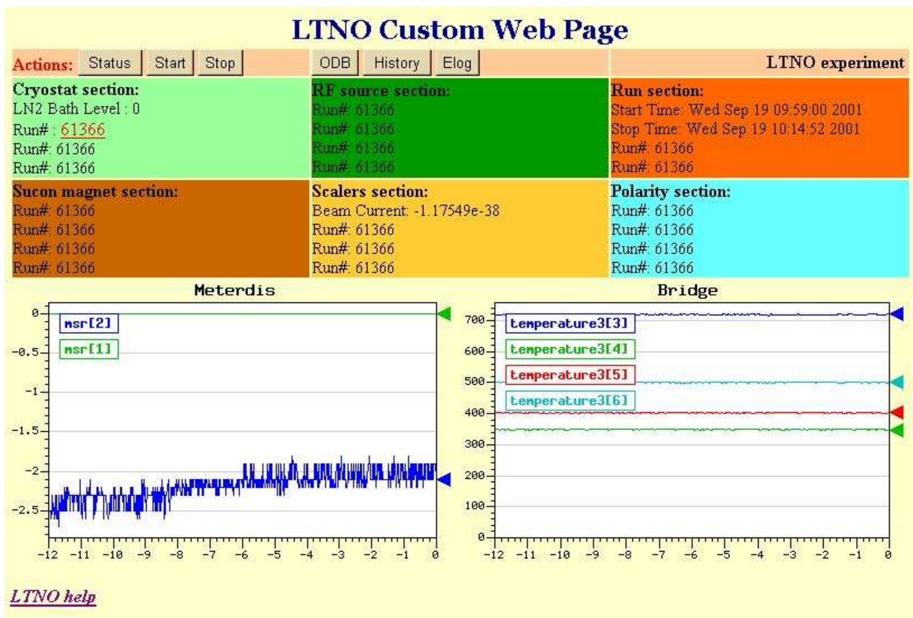


Figure 30: web page produced by mcustom.html.

Utilities - Top - Data format

6.13 New Documented Features

Top - Top - Introduction

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

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

- **Current doc revision: 1.9.5-0**

- **Software version: 1.9.5**
- **Latest tarball :** 1.9.5
- **Latest RPM :** 1.9.2-1
- **[1.9.5]**
  - **When upgrading to 1.9.5, *ALL* midas applications including user applications needs to be rebuild *AND* the ODB.SHM (.ODB.SHM) shared memory need to be removed. Prior the removal of the ODB.SHM, the ODB database can be saved in ASCII format for later restoration.**
  - **Run Transition Sequence changed to multiple level scheme.**
  - **odbedit_task support of XML format for ODB dump.**
  - **Large File support (>2GB) from mlogger task application.**
  - **Folder Root Histogram support within mana.**
  - **mevb task application.**
  - **New Midas Frontend application argument for Event Builder option (-i index).**
    - **Documentation on "Tests" results from analyzer.**
  - **mySQL support from mlogger task.**
  - **Increase system wide parameters values (see midas.h).**
  - **Fix numerous small annoying bugs...**
  - **Improve debugging messages in mserver -d (/tmp/mserver.log).**
- **[<1.9.5]**
  - **In writing**
    - **Epics Slow Control documentation**
- **Introduce MIDASSYS environment variable**
- **Analyzer documention revision MIDAS Analyzer**
- **Watchdog bug £x (RH9.0)**
- **Restructured Midas distribution**
  - **In the same effort as the documentation, the midas tree and CVS have been modi£ed. The** download area **now contains separate directories for doc, add-ons, publications etc.**

[DOCUMENTATION in progress]

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

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

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

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

- **[1.9.1]**
  - **This version addresses several bugs reported in the web interface, history, logger, odbedit and implements new features in particular for the history pages on web interface. The detail list of the modi£cations can be found in CHANGELOG .**
    - **[EQ_FRAGMENTED] Possibility to send extremely large event through the system without modi£cation of the system con£guration (see The Equipment structure)**
    - **[logger subdir option] Allows to redirect the data £les to a subdirectory based on the time of the creation of the data £le (see ODB /Logger Tree).**
    - **Option for building an analyzer without the CERN library (HBOOK) (see Midas build options and operation considerations).**
    - **[MOD. REQ.] This release requires several modi£cations in the user code in order to compile the 1.9.1.**
      1. **[db_get_value() function] Requires an extra parameter see Midas Code and Libraries.**
      2. **[max_event_size_frag] Required in all the frontend code as follow:**

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

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

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

Top - Top - Introduction

6.14 ODB Structure

Internal features - Top - Data format

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

Each ODB £eld 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 speci£c set of data. On the £rst invocation of the database (£rst 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 £rst time, the tree **/Run£nfo**, /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 ¤exibility 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 speci£c to each "Midas client" currenltly connected to the experiment. This information is not primarily 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 £elds 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]** Speci£es in which state the current run is. The possible states are 1: STOPPED, 2: RUNNING, 3: PAUSED.
- **[Online Mode]** Speci£es 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 ¤ag will be switched off.
- **[Run number]** Speci£es the current run number. This number is automatically incremented by a successful run start procedure.
- **[Transition in progress]** Speci£es the current internal state of the system. This parameter is used for multiple source of "run start" synchronization.

- **[Requested transition]** Speci£es the current internal of the Deferred Transition state of the system.
- **[Start Time]** Speci£es in an ASCII format the time at which the last run has been started.
- **[Start Time binary]** Speci£es in a binary format at the time at which the last run has been started This £eld is useful for time interval computation.
- **[Stop Time]** Speci£es in an ASCII format the time at which the last run has been stopped.
- **[Stop Time binary]** Speci£es in a binary format the time at which the last run has been stopped. This £eld 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 speci£c 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 de£ned. 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 £rst 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 is achieved by de£ning at build time the preprocessor ¤ag 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 prede£ned mySQL database name *Midas* and table *Runlog*. Under 2 dedicated sub directories i.e: BOR and EOR, prede£ned 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] Speci£es in which directory £les 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 £les, history £les, message £les. In the case of multiple logging channels, the data path for all the channels is defaulted to the same location. In the case where speci£c directory has to be assigned to each individual logging channel, the £eld **/logger/channel/<x>/Settings/Filename** can contain the full path of the location of the .mid, .ybs, .asc £le. By £nding the OS speci£c **SEPARATOR_DIR** ("/", "\"). The £eld **Filename** will overwite the global Data_Dir setting for that particular channel.
- [History Dir] This £eld is optional and doesn't appear by default in the logger. If present the location of the History system £les is reassigned to the de£ned path instead of the default Data_Dir .
- [Elog Dir] This £eld is optional and doesn't appear by default in the logger. If present the location of the Electronic Logbook £les is reassigned to the de£ned path instead of the default Data_Dir.
- [Message £le] Speci£es the £le name for the log £le which contains all messages from the MIDAS message system. The message log £le is a simple ASCII £le, which can be viewed at any time to see a history of what happened in an experiment.
- [Write data] Global ¤ag 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 prede£ned logger key for enabling data logging. This action can be overridden by setting the active key to 1.

- [ODB Dump] Speci£es if a dump of the complete ODB should be written to the £le speci£ed by ODB Dump File.
- [ODB Dump File] At the end of each run. If the £le name contains a "%", this gets replaced by the current run number similar to the printf() C function. The format speci£er 05d from above would be evaluated to a £ve digit run number with leading zeros like run00002.odb. The ODB dump £le 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().
- [Auto restart] When this ¤ag 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] Speci£es 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 de£ne 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 £le in case of a disk logging, where 05d is replaced by the current run number the same way as for the ODB dump £les. In the case of a tape logging, the £lename speci£es a tape device like /dev/nrmt0 or /dev/nst0 under UNIX or \\.\tape0 under Windows NT.
  - In FTP mode, the £lename speci£es 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] Speci£es the format to be used for writing the data to the logging channel. It can one of the £ve value: MIDAS, YBOS, ROOT, ASCII and DUMP. The MIDAS and YBOS binary formats Midas format and YBOS format, respectively. The extention for the £le 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 £le 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] Speci£es 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 re¤ecting 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-£eld for logging system messages. If a bit in this £eld 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 £lters 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 £ltered events get logged.

- [Event limit, Byte limit and Tape capacity] These £elds 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 £les on tape have to be skipped in order to reach a speci£c run.

- [Subdir format, Current £lename] In the case the **Subdir format** is not empty, this £eld will enable the placement of the data log £le into a sub directory. The name of this subdirectory is composed by the given **Subdir** format string. Its format follows the de£nition of the system call strftime() . Ordinary characters placed in the format string are copied to s without conversion. Conversion speci£ers 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 £lename] will re¤ect the full path of the saved data £le.

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 de£ned labeled **Name** for the experiment name. The user can create four system keys in order to provide extra run control ¤exibility i.e.: **"Run Parameter/"**, **"Edit on Start/"**, **"Lock when running/"** and **"Security/"**.

```
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] Speci£es the name of the experiment.
- [Run Parameters] Speci£es a £x directory name where you can create and de£ne 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" de£ned 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] Speci£es a £x directory name where you can de£ne an ODB link (similar to a symbolic link in UNIX) key to the pre-de£ned 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 de£ning 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] Speci£es a £x directory for de£ning 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 modi£cation 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] Speci£es a £x 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 de£ned.

- [Password] Speci£es 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-£elds will be present under the Security tree.

  - Password
  - Allowed hosts
  - Allowed programs
  - Web Password

- [Allowed hosts] Speci£es a £x directory name where allowed remote hostname can be de£ned 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 con£guration). To address this problem speci£c 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] Speci£es 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] Speci£es a separate password for the Web server access (mhttpd task). If this £eld 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 de£ne 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 £elds.

Where the 2 trigger £elds are symbolic links to the given path. The sub-tree **System** de£nes 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 de£ne the condition to be tested and the **"Classes"** which de£nes the action to be taken when the alarm occurs. In order to make the system ¤exible, 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 ¤ag.
- [Alarms] Sub-tree de£ning each individual alarm condition.
- [Classes] Sub-tree de£ning each individual action to be performed by a pre-de£ned 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 £eld (or links). The £rst ODB £eld 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] De£nes 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-de£ned types displayed when composing an Elog entry. A maximum of 20 types are available. The list will be terminated by the encounter of the £rst blank type.
- [Systems] Pre-de£ned categories displayed when composing an Elog entry. A maximum of 20 types are available. The list will be terminated by the encounter of the £rst 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 speci£c 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 £rst activation of lazylogger task. This task connects to a de£ned 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

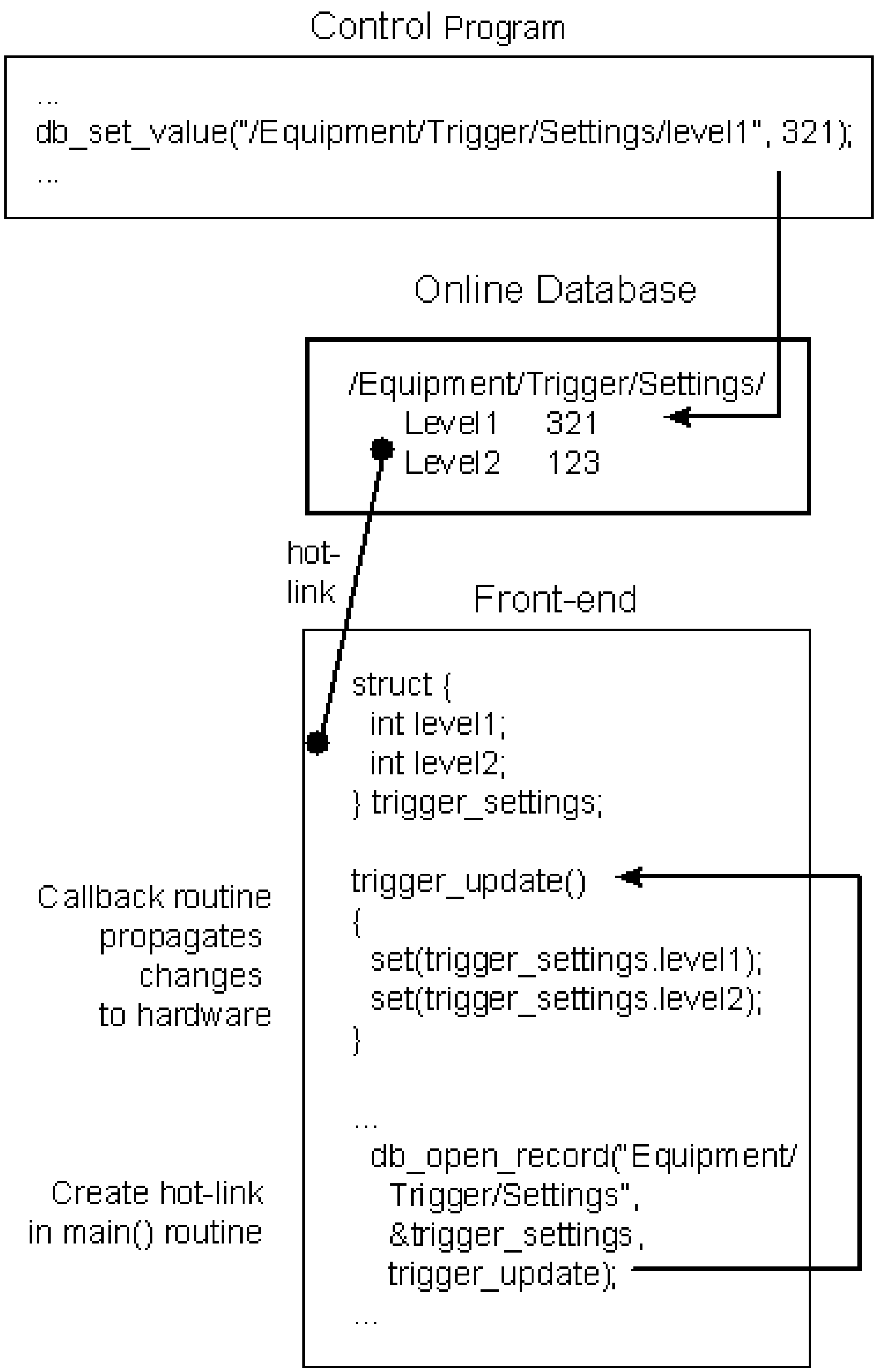


Figure 31: HotLink process

First the parameters have to be de£ned 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 de£ne 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 £le containing this C structure. This £le 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 £le **experim.h** :

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

Now this £le 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 de£nition can be used to de£ne 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 modi£ed, 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 modi£cation the frontend should display the new values. The settings can be saved to a £le 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 modi£ed from any application just by accessing the ODB. Following listing is a complete user application that modi£es 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 £gure 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 £le" like for later retrieval. One set of £le 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 de£nition 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 £eld value is different then zero. The value will re¤ect the periodicity of the history logging (see The Equipment structure).

- **"Virtual History event"**: Composed within the Online Database under the speci£c tree "/History/Links" (see ODB /History Tree)

Both de£nition mode takes effects when the data logger gets a "start run" transition. Any modi£cation 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 £les using the de£nition 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
          ...
```

- [£xed event] The names of the individual element under **/equipment/**<...>/variables/ will be used for the history name composition.
- [£xed 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]

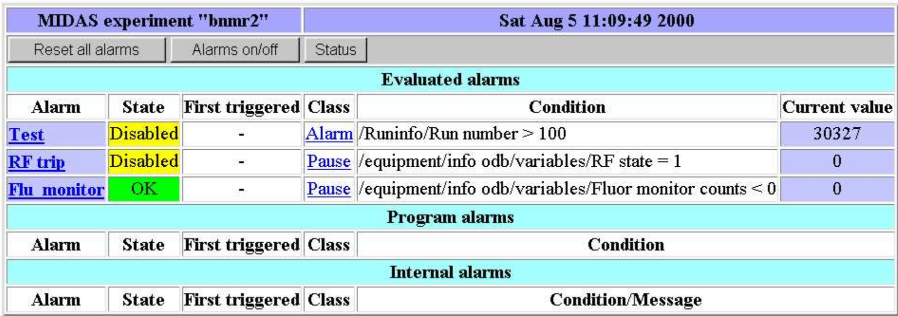
6.14.17 Alarm System

The alarm system is built in and part of the main experiment scheduler. This means no separate task is necessary to bene£cate from it, but this feature is active during **ONLINE** mode **ONLY** . Alarm setup and activation is done through the Online DataBase. Alarm system includes several other features such as: sequencing control of the experiment. The alarm capabilities are:

- Alarm setting on any ODB variables against threshold parameter.
- Alarm check frequency
- Alarm trigger frequency
- Customizable alarm scheme, under this scheme multiple choice of alarm type can be selected.
- Program control on run transition.

Beside the setup through ODBEdit, the Alarm can also be setup through the Midas web page..

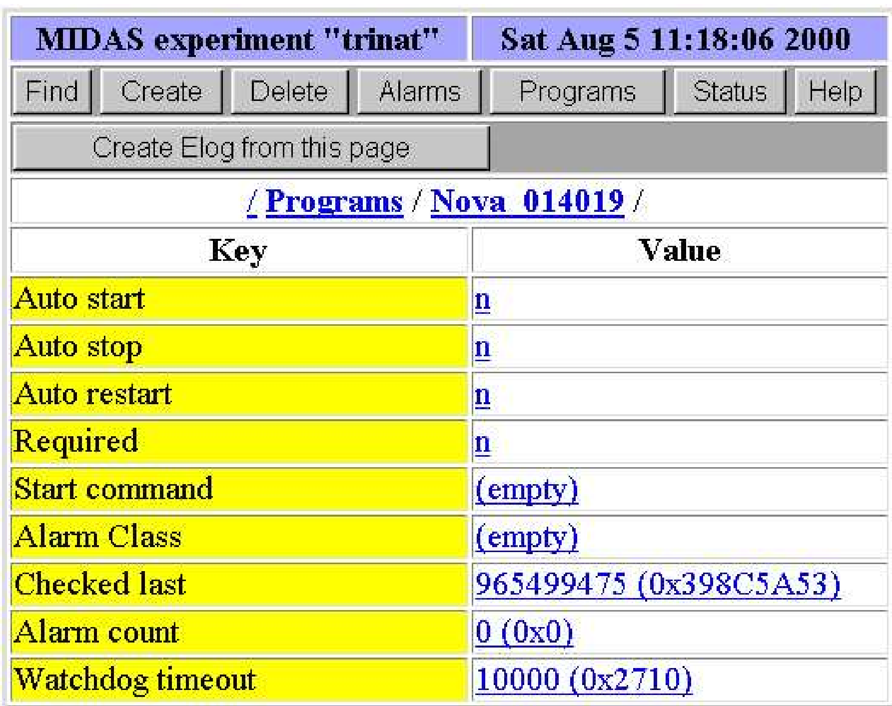
Midas Web Alarm setting display



MIDAS experiment "bnmr2"				Sat Aug 5 11:09:49 2000	
Reset all alarms	Alarms on/off	Status			
Evaluated alarms					
Alarm	State	First triggered	Class	Condition	Current value
Test	Disabled	-	Alarm	/Runinfo/Run number > 100	30327
RF trip	Disabled	-	Pause	/equipment/info odb/variables/RF state = 1	0
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 32: Midas Web Alarm setting display

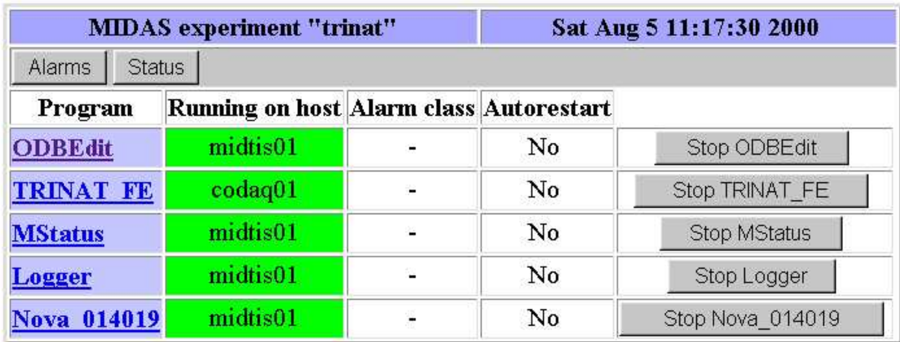
Midas Web Alarm setting display



MIDAS experiment "trinat"	Sat Aug 5 11:18:06 2000
Find Create Delete Alarms Programs Status Help	
Create Elog from this page	
/ Programs / Nova_014019 /	
Key	Value
Auto start	n
Auto stop	n
Auto restart	n
Required	n
Start command	(empty)
Alarm Class	(empty)
Checked last	965499475 (0x398C5A53)
Alarm count	0 (0x0)
Watchdog timeout	10000 (0x2710)

Figure 33: 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 34: 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 re¤ect the latest installation and basic operation of the Midas package.

... This section will... describes step-by-step the installation procedure of the Midas package on several platform as well as the procedure to run a demo sample experiment. In a second stage, the frontend or the analyzer can be moved to another computer to test the remote connection capability.

The Midas Package source and binaries can be found at : `PSI` or at `TRIUMF` . An `online CVS 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 £les** The compressed £le 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 £les or the CVS repository.** In the case of the **rpm**, the binaries are placed in the **/usr/local/bin /usr/local/include /usr/local/lib**.
- **CVS** The source code can be extracted from the CVS repository. The following two anonymous commands can used for respectively **checking out** (£rst time) and **updating** the full midas tree. The password required for access is "cvs".

```
cvs -e ssh -d :ext:cvs@midas.psi.ch:/usr/local/cvsroot checkout midas

cvs -e ssh -d :ext:cvs@midas.psi.ch:/usr/local/cvsroot update
```

2. **Installation:** The installation consists in placing the image £les in the **/usr/local/** directories. This operation requires superuser privilege. The open "install" from the Make£le will automatically do this installation for you.

```
cd /home/mydir/midas
su -
make install
```

3. **Con£guration:** Several system £les needs to be modi£ed 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 de£nition. Create new £le 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 £le (add /usr/local/lib).

```
/usr/local/lib
```

The system environment **LD_LIBRARY_PATH** can used instead.

- **/etc/exptab :** Midas Experiment de£nition £le (see below).

4. **Experiment de£nition:** Midas system supports multiple experiment running contemporary on the same computer. Even though it may not be ef£cient, 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 identi£ed set of parameter for each experiment that is used to de£ne 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 speci£ed 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 £le **exptab**. This £le **exptab** is expected by default to be located under **/etc/exptab**. This can be overwritten by the Environment variables MIDAS_EXPTAB.

   exptab £le is composed of one line per experiment de£nition. 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 de£ned into **exptab** are not accessible remotely, but can still be accessed locally using the Environment variables MIDAS_DIR if de£ned. This environment superceed the **exptab** de£nition.

5. **Compilation & Build:** You should be able to rebuild the full package once the Midas tree structure has been placed in your temporary directory. The compilation and link will try to generate the **rmidas** application which requires **ROOT**. The application **mana** will also be compiled for **HBOOK** and **ROOT**. Look in the make listing below for the HAVE_HBOOK, HAVE_ROOT.

```
> cd /home/mydir/midas
> make
cc -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/lib/midas.o src/midas.c
cc -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/lib/system.o src/system.c
cc -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/lib/mrpc.o src/mrpc.c
cc -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
```

```
-o linux/lib/odb.o src/odb.c
cc -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/lib/ybos.o src/ybos.c
cc -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/lib/ftplib.o src/ftplib.c
rm -f linux/lib/libmidas.a
ar -crv linux/lib/libmidas.a linux/lib/midas.o linux/lib/system.o linux/lib/mrpc.o
linux/lib/odb.o linux/lib/ybos.o linux/lib/ftplib.o
a - linux/lib/midas.o
a - linux/lib/system.o
a - linux/lib/mrpc.o
a - linux/lib/odb.o
a - linux/lib/ybos.o
a - linux/lib/ftplib.o
rm -f linux/lib/libmidas.so
ld -shared -o linux/lib/libmidas.so linux/lib/midas.o linux/lib/system.o
linux/lib/mrpc.o linux/lib/odb.o linux/lib/ybos.o linux/lib/ftplib.o -lutil
-lpthread -lc
cc -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/lib/mana.o src/mana.c
cc -Dextname -DHAVE_HBOOK -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib
-DINCLUDE_FTPLIB -DOS_LINUX -fPIC    -o linux/lib/hmana.o src/mana.c
...
g++ -DHAVE_ROOT -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB
-DOS_LINUX -fPIC   -D_REENTRANT -I/home1/midas/ root/include -o linux/lib/rmana.o
src/mana.c
g++ -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINU
-fPIC   -o linux/lib/mfe.o src/mfe.c
cc -Dextname -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib
-DINCLUDE_FTPLIB -DOS_LINUX -fPIC    -o linux/lib/fal.o src/fal.c
...
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mserver src/mserver.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mhttpd src/mhttpd.c src/mgd.c -lmidas -lutil -lpthread -lm
g++ -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-DHAVE_ROOT -D_REENTRANT -I/home1/midas/root/include
-o linux/bin/mlogger src/mlogger.c -lmidas
-L/home1/midas/root/lib -lCore -lCint -lHist -lGraf -lGraf3d -lGpad -lTree
-lRint -lPostscript -lMatrix -lPhysics -lpthread -lm -ldl -rdynamic -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/odbedit src/odbedit.c src/cmdedit.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mtape utils/mtape.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mhist utils/mhist.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mstat utils/mstat.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mcnaf utils/mcnaf.c drivers/bus/camacrpc.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mdump utils/mdump.c -lmidas -lz -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/lazylogger src/lazylogger.c -lmidas -lz -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mchart utils/mchart.c -lmidas -lutil -lpthread
cp -f utils/stripchart.tcl linux/bin/.
```

```
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/webpaw utils/webpaw.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/odbhist utils/odbhist.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/melog utils/melog.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mlxspeaker utils/mlxspeaker.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/dio utils/dio.c -lmidas -lutil-lpthread
g++ -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-DHAVE_ROOT -D_REENTRANT -I/home1/midas/root/include -o linux/bin/rmidas src/rmidas.c
-lmidas -L/home1/midas/root/lib -lCore -lCint -lHist -lGraf -lGraf3d -lGpad
-lTree -lRint -lPostscript -lMatrix -lPhysics -lGui -lpthread -lm -ldl -rdynamic
-lutil -lpthread
```

6. **Demo examples:** The midas £le structure contains examples of code which can be (should be) used for template. In the **midas/examples/experiment** you will £nd a full set for frontend and analysis code. The building of this example is performed with the **Make£le** 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 Make£le

```
#-------------------------------------------------------------------
# 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. **Con£guration:**

4. **Experiment de£nition:**

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 £le : 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 the hardware and transmit this information to a central place where data logging and analysis can be performed. This task is achieved with a) a speci£c code written by the user describing the sequence of action to acquire the hardware data and b) a framework code handling the data ¤ow control, data transmission and run control operation. From Midas version 1.9.5 a new argument (-i index) has been introduce to facilitate the multiple frontend con£guration 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] :Speci£es host to connect to. Must be a valid IP host name. This option supersedes the MIDAS_SERVER_HOST environment variable.
  - [-e exptname] :Speci£es 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 £les.
  - [-c @commandFile] :Perform commands in sequence found in the commandFile.
  - [-s size] : size in byte (for creation). Specify the size of the ODB £le to be created when no share £le 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 character left from cursor
  - [Delete/Ctrl-D] Erase character under cursor
  - [Ctrl-W/Ctrl-U] Erase current line

- [Ctrl-K] Erase 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 £le system. Most commands are similar to UNIX £le 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 £le 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. Speci£es 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 / of¤ine 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 <£lename1> <£lename2> Con£guration £le name(s). May contain a '%05d' to be replaced by the run number. Up to ten £les can be speci£ed in one "-c" statement.
  - -d Debug ¤ag when started the analyzer fron 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 £le only if analyzer accepts them (doesn't return ANA_SKIP).
  - -h <hostname> MIDAS host to connect to when running the analyzer online (see odbedit task)
  - -i <£lename1> <£lename2> Input £le name. May contain a '%05d' to be replaced by the run number. Up to ten input £les can be speci£ed in one "-i" statement.
  - -l If set, don't load histos from last histo £le when running online.
  - -L HBOOK LREC size. Default is 8190.
  - -n <count> Analyze only "count" events.
  - -n <£rst> <last> Analyze only events from "£rst" to "last".
  - -n <£rst> <last> <n> Analyze every n-th event from "£rst" to "last".
  - -o <£lename> Output £le name. Extension may be .mid (MIDAS binary), .asc (ASCII) or .rz (HBOOK). If the name contains a '%05d', one output £le is generated for each run. Use "OFLN" as output £le name to creaate a HBOOK shared memory instead of a £le.

- -p <param=value> Set individual parameters to a speci£c value. Overrides any setting in con£guration £les
- -P <ODB tree> Protect an ODB subtree from being overwritten with the online data when ODB gets loaded from .mid £le
- -q Quiet ¤ag. If set, don't display run progress in of¤ine mode.
- -r <range> Range of run numbers to analyzer like "-r 120 125" to analyze runs 120 to 125 (inclusive). The "-r" ¤ag must be used with a '%05d' in the input £le 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 £le. 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 frontend under certain condition and store them onto physical device such as *disk* or *tape*. It also act as an history event collector if either the history ¤ags is enabled in the frontend equipment (see The Equipment structure or if the ODB tree /History/Links is de£ned (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 is running, the history is 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 de£ned while running but effect will be taken place only at the following begin of run transition.
  - The default setting de£nes a data "Midas" format with a £le 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 £le size limitation (<2GB) has been removed for older OS version.
    - mySQL data entry support.

6.15.8 lazylogger task

lazylogger is an application which decouples the data aquisition from the data logging mechanism. The need of such application has been dictated by the slow response time of some of the media logging devices (Tape devices). Delay due to tape mounting, retension, reposition imply that the data acquisition has to hold until operation completion. By using **mlogger** to log data to disk in a £rst stage and then using **lazylogger** to copy or move the stored £les to the "slow device" we can keep the acquisition running without interruption.

- Multiple lazylogger can be running comtemporary on the same computer, each one taking care of a particular channel.
- Each lazylogger channel 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 manipulate by the Lazylogger channel. This list won't exists until the £rst successful operation of the channel is complete.

- **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 de£ned 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 will contains all the recorded data £les. 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 £le is de£ned as:

      The data £le 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 de£nes how many consecutive data £le 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 logger 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 new run 254 will start. -3 "Stay behind = -3" will correspond to 3 £le untouch on the disk (#251, #252, #253). The negative sign instructs lazylogger to **always** scan the entire "Data Dir" from the oldest to the most recent £le 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 older 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 lazylogger should be actived. By default lazylogger is **ALWAYS-** running. In the case of high data rate acquisition it could be necessary to activate lazylogger only when the run is either pauses, stopped or when some external condition is satis£ed such as "Low beam intensity". In this later case, condition based on a single £eld 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 £les. 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 £les. Currently supported format are: **MIDAS** or **YBOS**.

– **[Filename format]** Specify the £le format of the data £les. 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 be run after completion of a lazylogger backup set (see below "Capacity (Bytes)").

– **[Path]** Specify the "slow device" path. Three possible type 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, lazylogger will close the backup device and clear the "List Label" £eld 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 £les to the "slow device". This label is used only internaly by 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 £eld 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 £les have been removed from the disk.
  - **[Exec preW £le]** Permits to run a script before the begining of the lazy job. The **Arguments** passed to the scripts are: input £le name , output £le name, current block number.
  - **[Exec postW £le]** 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, £le name, £le size in MB, current block number.
  - **[Modulo.Position]** This £eld is for multiple instance of 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 instance are simultaneously running (3.) and the position of which this instance is assigned to (.1). As an example for 3 lazylogger running contemporaneously the £eld 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 ¤ag is not enable 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 to the array name backup-set label. Any run number appearing in any of the array is considered to have been backed up.

- **Usage** lazylogger requires to be setup prior data £le can be moved. This setup consists in 4 steps:

- **[Step 1]** Invoking 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 £eld 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 de£ned 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 operate 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 produce 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 864.02
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 on producing a log message informing about the actual percentage of the backup being 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 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 £le. Speci£c 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 assemble a new overall Midas event.

In the case where overall data collection is handled by multiple physically separated frontend, 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 speci£c 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 install prior 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 allow "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 mailto: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 £rst. 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 de£ned 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 speci£ed Elog port.

- **Arguments**
  - [-h] : help
  - [-h hostname] : host name
  - [-l exptname or logbook]
  - [-u username password]
  - [-f <attachment>] : up to 10 £les.
  - -a <attribute>=<value> : up to 20 attributes. The attribute "Author=..." must at least be present for submission of Elog.
  - -m <text£le> | text> Arguments with blanks must be enclosed in quotes. The elog message can either be submitted on the command line or in a £le with the -m ¤ag. 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" £eld the full speci£cation of the attribute. In case of multiple attachement, only one "-f" is required followed by up to 10 £le 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 £le 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 display through the Midas web page (see mhttpd task).

- **Example**

Midas Web History display.

Figure 35: Midas Web History display.

6.15.15 mchart task

mchart is a periodic data retriever of a speci£c path in the ODB which can be used in conjunction with a stripchart graphic program.

- In the £rst of two step procedure, a speci£c path in the ODB can be scanned for composing a con£guration £le by extracting all numerical data references **£le.conf** .
- In the second step the mchart will produce at £x time interval a refreshed data £le containing the values of the numerical data speci£ed in the con£guration £le. This £le is then available for a stripchart program to be used for chart recording type of graph.

Two possible stripchart available are:

- **gstripchart** The con£guration £le generated by mchart is compatible with the GNU stripchart which permit so£sticated data equation manipulation. In the other hand, the data display is not very fency and provide just a basic chart recorder.
- stripchart.tcl £le 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 £le] : £le name (+.conf: if using existing £le).
  - [-q ODBpath] : ODB tree path for extraction of the variables.
  - [-c] : ONLY creates the con£guration £le 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 con£guration contains an entry for each variable found under the ODBpath requested. The format is described in the gstripchart documentation.

Once the con£guration £le has been created, it is possible to apply any valid operation (equation) to the parameters of the £le following the gstripchart syntax.

In the case of the use of the *stripchart* from G.Hofman, only the "£lename", "pattern", "maximum", "minimum" £elds are used.

When using mchart with -D Argument, it is necessary to have the MCHART_DIR de£ned in order to allow the daemon to £nd the location of the con£guration and data £les (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 £le (data £le) will be updated a £xed 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 £le (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 con£guration £le 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 con£guration £le 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 Make£le.

```
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 allows you to gain access to 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 port from 0x200 to 0x3FF.
- As this mode of I/O access by-passes the driver (if any), concurrent access to the same I/O port may produce unexpected result and in the worth case 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 £le

Graphical stripchart data display. Operates on mchart task data or on Midas history save-set £les. (see also History system).

- **Arguments**
  - [-mhist] : start stripchart for Midas history data.
- **Usage** : stripchart <-options> <con£g-£le> -mhist (look at history £le -default) -dmhist debug mhist -debug debug stripchart con£g_£le: see mchart_task

```
> stripchart.tcl -debug
> stripchart.tcl
```

- **Example**

```
> stripchart.tcl -h
```

gstripchart display with parameters and data pop-up.

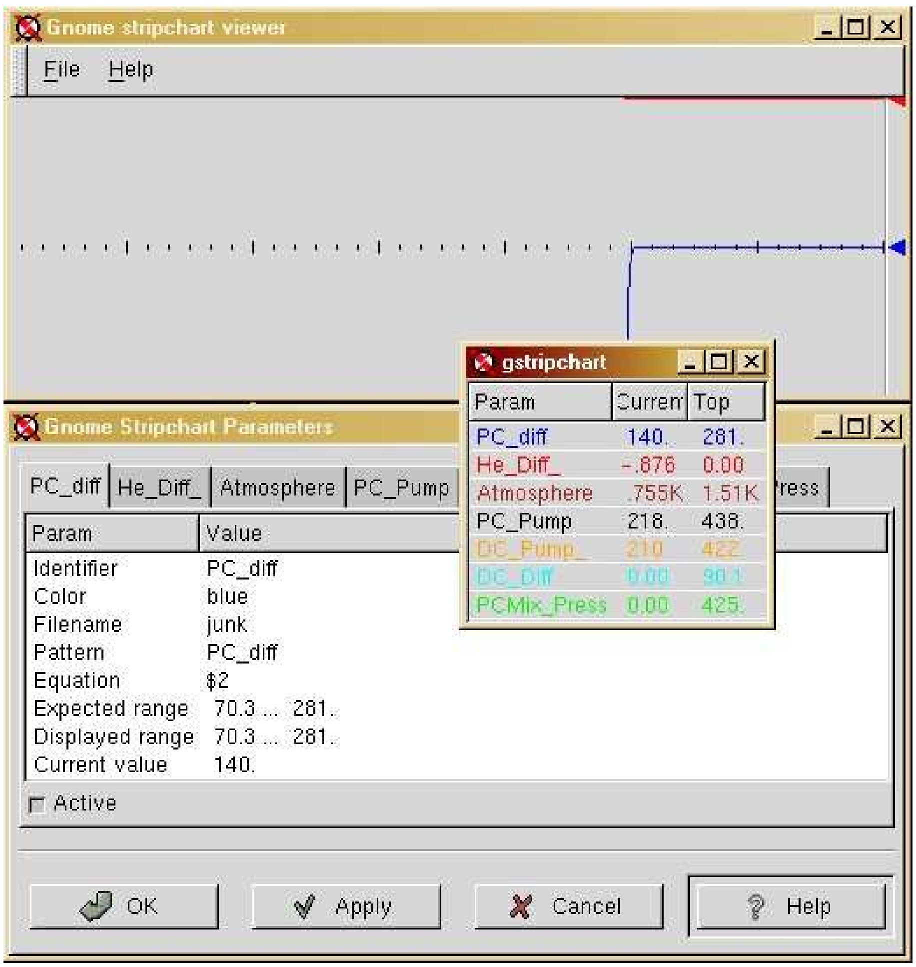


Figure 36: gstripchart display with parameters and data pop-up.

stripchart.tcl mhist mode: main window with pull-downs.

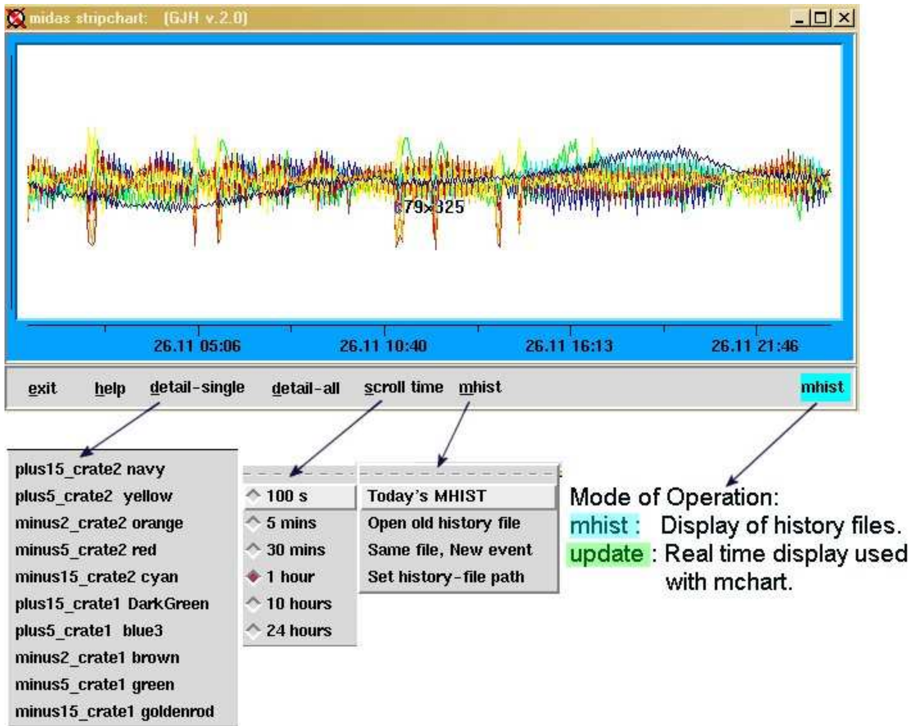


Figure 37: stripchart.tcl mhist mode: main window with pull-downs.

stripchart.tcl Online data, running in conjunction with mchart

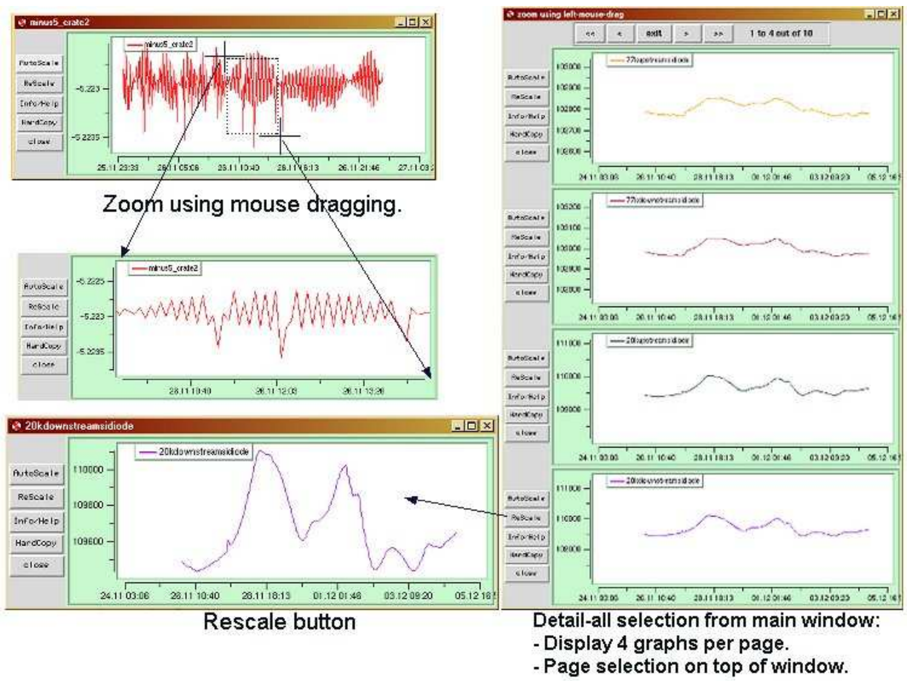


Figure 38: stripchart.tcl Online data, running in conjunction with mchart

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.

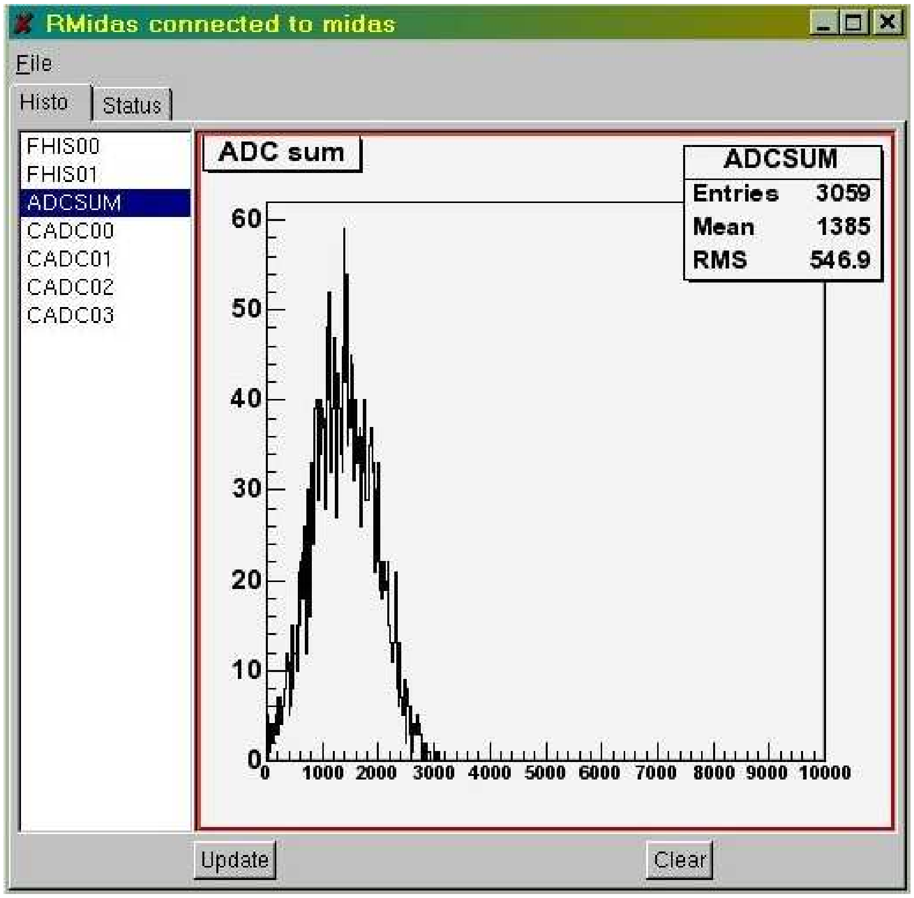


Figure 39: rmidas display sample. Using the example/experiment/ demo setup.

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 between 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 not to be modi£ed. In the case debugging is required, the *mserver* can be started with the -d ¤ag which will write an entry for each transaction appearing into 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**