

Midas Reference Manual  
1.9.3-1

Generated by Doxygen 1.3.5

Wed Jan 14 22:05:10 2004



# Contents

<b>1</b>	<b>Midas documentation</b>	<b>1</b>
1.1	Introduction . . . . .	1
<b>2</b>	<b>Midas Module Index</b>	<b>3</b>
2.1	Midas Modules . . . . .	3
<b>3</b>	<b>Midas Data Structure Index</b>	<b>5</b>
3.1	Midas Data Structures . . . . .	5
<b>4</b>	<b>Midas File Index</b>	<b>7</b>
4.1	Midas File List . . . . .	7
<b>5</b>	<b>Midas Page Index</b>	<b>9</b>
5.1	Midas Related Pages . . . . .	9
<b>6</b>	<b>Midas Module Documentation</b>	<b>11</b>
6.1	Midas VME Standard . . . . .	11
6.2	VME #define . . . . .	12
6.3	VME Functions (vme_xxx) . . . . .	17
6.4	Midas CAMAC standard . . . . .	20
6.5	Camac Functions (camxxx) . . . . .	21
6.6	The midas.h & midas.c . . . . .	43
6.7	Midas #define . . . . .	53
6.8	Midas Macros . . . . .	72
6.9	Midas Error definition . . . . .	73

6.10 Midas Structure Declaration . . . . .	74
6.11 Status and error codes . . . . .	75
6.12 Buffer Manager error codes . . . . .	78
6.13 Online Database error codes . . . . .	82
6.14 System Services error code . . . . .	88
6.15 Remote Procedure Calls error codes . . . . .	94
6.16 Other errors . . . . .	97
6.17 Buffer Section . . . . .	106
6.18 Equipment related . . . . .	109
6.19 Bank related . . . . .	111
6.20 Analyzer related . . . . .	112
6.21 History related . . . . .	113
6.22 ODB runinfo related . . . . .	114
6.23 Alarm related . . . . .	115
6.24 The ybos.h & ybos.c . . . . .	119
6.25 YBOS #define . . . . .	120
6.26 YBOS Macros . . . . .	126
6.27 YBOS error code . . . . .	130
6.28 YBOS Bank Functions (ybk_xxx) . . . . .	132
6.29 Midas Common Functions (cm_xxx) . . . . .	136
6.30 Midas Buffer Manager Functions (bm_xxx) . . . . .	154
6.31 Midas Message Functions (msg_xxx) . . . . .	165
6.32 Midas Bank Functions (bk_xxx) . . . . .	171
6.33 Midas Alarm Functions (al_xxx) . . . . .	176
6.34 Midas History Functions (hs_xxx) . . . . .	177
6.35 Midas Elog Functions (el_xxx) . . . . .	178
6.36 Midas RPC Functions (rpc_xxx) . . . . .	180
6.37 Midas Dual Buffer Memory Functions (dm_xxx) . . . . .	183
6.38 System Functions (ss_xxx) . . . . .	184
6.39 The msystem.h & system.c . . . . .	189
6.40 System #define . . . . .	190

---

6.41	System Macros . . . . .	192
6.42	System Structure Declaration . . . . .	194
6.43	The mrpc.h & mrpc.c . . . . .	195
6.44	RPC #define . . . . .	196
6.45	Midas RPC_LIST . . . . .	213
6.46	The odb.c . . . . .	215
6.47	Midas ODB Functions (db_XXX) . . . . .	216
<b>7</b>	<b>Midas Data Structure Documentation</b>	<b>245</b>
7.1	ALARM Struct Reference . . . . .	245
7.2	ALARM_CLASS Struct Reference . . . . .	247
7.3	ANA_MODULE Struct Reference . . . . .	249
7.4	ANA_TEST Struct Reference . . . . .	251
7.5	ANALYZE_REQUEST Struct Reference . . . . .	252
7.6	AR_INFO Struct Reference . . . . .	256
7.7	AR_STATS Struct Reference . . . . .	258
7.8	BANK Struct Reference . . . . .	259
7.9	BANK32 Struct Reference . . . . .	260
7.10	BANK_HEADER Struct Reference . . . . .	261
7.11	BANK_LIST Struct Reference . . . . .	262
7.12	BUFFER Struct Reference . . . . .	264
7.13	BUFFER_CLIENT Struct Reference . . . . .	268
7.14	BUFFER_HEADER Struct Reference . . . . .	272
7.15	BUS_DRIVER Struct Reference . . . . .	275
7.16	DATABASE Struct Reference . . . . .	276
7.17	DATABASE_CLIENT Struct Reference . . . . .	278
7.18	DATABASE_HEADER Struct Reference . . . . .	280
7.19	DEF_RECORD Struct Reference . . . . .	282
7.20	DEVICE_DRIVER Struct Reference . . . . .	283
7.21	eqpmnt Struct Reference . . . . .	284
7.22	EQUIPMENT_INFO Struct Reference . . . . .	288

---

7.23 EQUIPMENT_STATS Struct Reference . . . . .	291
7.24 EVENT_HEADER Struct Reference . . . . .	292
7.25 EVENT_REQUEST Struct Reference . . . . .	294
7.26 FREE_DESCRIP Struct Reference . . . . .	296
7.27 HIST_RECORD Struct Reference . . . . .	297
7.28 HISTORY Struct Reference . . . . .	298
7.29 INDEX_RECORD Struct Reference . . . . .	299
7.30 KEY Struct Reference . . . . .	300
7.31 KEYLIST Struct Reference . . . . .	303
7.32 OPEN_RECORD Struct Reference . . . . .	304
7.33 PROGRAM_INFO Struct Reference . . . . .	305
7.34 RECORD_LIST Struct Reference . . . . .	307
7.35 REQUEST_LIST Struct Reference . . . . .	309
7.36 RUNINFO Struct Reference . . . . .	310
7.37 TAG Struct Reference . . . . .	312
 <b>8 Midas File Documentation</b>	 <b>313</b>
8.1 analyzer.dox File Reference . . . . .	313
8.2 appendixA.dox File Reference . . . . .	314
8.3 appendixB.dox File Reference . . . . .	315
8.4 appendixC.dox File Reference . . . . .	316
8.5 appendixD.dox File Reference . . . . .	317
8.6 appendixE.dox File Reference . . . . .	318
8.7 appendixG.dox File Reference . . . . .	319
8.8 components.dox File Reference . . . . .	320
8.9 ebuser.c File Reference . . . . .	321
8.10 esone.c File Reference . . . . .	324
8.11 eventbuilder.dox File Reference . . . . .	337
8.12 internal.dox File Reference . . . . .	338
8.13 introduction.dox File Reference . . . . .	339
8.14 mcstd.h File Reference . . . . .	340

---

8.15	mevb.c File Reference . . . . .	343
8.16	mfe.c File Reference . . . . .	351
8.17	mhttpd.dox File Reference . . . . .	359
8.18	midas.c File Reference . . . . .	360
8.19	midas.dox File Reference . . . . .	363
8.20	midas.h File Reference . . . . .	364
8.21	mrpc.c File Reference . . . . .	371
8.22	mrpc.h File Reference . . . . .	372
8.23	msystem.h File Reference . . . . .	375
8.24	mvmestd.h File Reference . . . . .	376
8.25	newdocfeatures.dox File Reference . . . . .	377
8.26	odb.c File Reference . . . . .	378
8.27	odbstruct.dox File Reference . . . . .	380
8.28	quickstart.dox File Reference . . . . .	381
8.29	system.c File Reference . . . . .	382
8.30	utilities.dox File Reference . . . . .	383
8.31	ybos.c File Reference . . . . .	384
8.32	ybos.h File Reference . . . . .	385
<b>9</b>	<b>Midas Page Documentation</b>	<b>387</b>
9.1	MIDAS Analyzer . . . . .	387
9.2	Data format . . . . .	389
9.3	Supported hardware . . . . .	395
9.4	CAMAC and VME access function call . . . . .	399
9.5	Midas build options and operation considerations . . . . .	402
9.6	Midas Code and Libraries . . . . .	407
9.7	Frequently Asked Questions . . . . .	416
9.8	Components . . . . .	419
9.9	Event Builder Functions . . . . .	429
9.10	Internal features . . . . .	434
9.11	Introduction . . . . .	457

---

9.12 mhttpd task . . . . .	459
9.13 New Documented Features . . . . .	481
9.14 ODB Structure . . . . .	484
9.15 Quick Start . . . . .	512
9.16 Utilities . . . . .	518



# Chapter 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**(p.??) : Whats new in Midas.
- **Introduction**(p.??) : Some intial words and description
- **Components**(p.??) : listing
- **Quick Start**(p.??) : The HowTo for installation.
- **Internal features**(p.??) : The main internal components of the system
- **Utilities**(p.??) : The Midas applications.
- **Data format**(p.??) : Supported data format
- **Supported hardware**(p.??) : Supported hardware.
- **CAMAC and VME access function call**(p.??) : CAMAC and VME access function call.

- **Midas build options and operation considerations**(p.??) : Midas build options and operation consideration.
- **Midas Code and Libraries**(p.??) : Midas Library.
- **Frequently Asked Questions**(p.??) : Frequently Asked Questions.

## Chapter 2

# Midas Module Index

### 2.1 Midas Modules

Here is a list of all modules:

Midas VME Standard . . . . .	??
VME #define . . . . .	??
VME Functions (vme_xxx) . . . . .	??
Midas CAMAC standard . . . . .	??
Camac Functions (camxxx) . . . . .	??
The midas.h & midas.c . . . . .	??
Midas #define . . . . .	??
Midas Macros . . . . .	??
Midas Error definition . . . . .	??
Status and error codes . . . . .	??
Buffer Manager error codes . . . . .	??
Online Database error codes . . . . .	??
System Services error code . . . . .	??
Remote Procedure Calls error codes . . . . .	??
Other errors . . . . .	??
Midas Structure Declaration . . . . .	??
Buffer Section . . . . .	??
Equipment related . . . . .	??
Bank related . . . . .	??
Analyzer related . . . . .	??
History related . . . . .	??
ODB runinfo related . . . . .	??
Alarm related . . . . .	??
Midas Message Functions (msg_xxx) . . . . .	??

Midas Common Functions (cm_xxx) . . . . .	??
Midas Buffer Manager Functions (bm_xxx) . . . . .	??
Midas RPC Functions (rpc_xxx) . . . . .	??
Midas Bank Functions (bk_xxx) . . . . .	??
Midas History Functions (hs_xxx) . . . . .	??
Midas Elog Functions (el_xxx) . . . . .	??
Midas Alarm Functions (al_xxx) . . . . .	??
Midas Dual Buffer Memory Functions (dm_xxx) . . . . .	??
The ybos.h & ybos.c . . . . .	??
YBOS #define . . . . .	??
YBOS error code . . . . .	??
YBOS Macros . . . . .	??
YBOS Bank Functions (ybk_xxx) . . . . .	??
The msystem.h & system.c . . . . .	??
System Functions (ss_xxx) . . . . .	??
System #define . . . . .	??
System Macros . . . . .	??
System Structure Declaration . . . . .	??
The mrpc.h & mrpc.c . . . . .	??
RPC #define . . . . .	??
Midas RPC_LIST . . . . .	??
The odb.c . . . . .	??
Midas ODB Functions (db_xxx) . . . . .	??

## Chapter 3

# Midas Data Structure Index

### 3.1 Midas Data Structures

Here are the data structures with brief descriptions:

ALARM	??
ALARM_CLASS	??
ANA_MODULE	??
ANA_TEST	??
ANALYZE_REQUEST	??
AR_INFO	??
AR_STATS	??
BANK	??
BANK32	??
BANK_HEADER	??
BANK_LIST	??
BUFFER	??
BUFFER_CLIENT	??
BUFFER_HEADER	??
BUS_DRIVER	??
DATABASE	??
DATABASE_CLIENT	??
DATABASE_HEADER	??
DEF_RECORD	??
DEVICE_DRIVER	??
eqpmnt	??
EQUIPMENT_INFO	??
EQUIPMENT_STATS	??
EVENT_HEADER	??
EVENT_REQUEST	??

FREE_DESCRIP . . . . .	??
HIST_RECORD . . . . .	??
HISTORY . . . . .	??
INDEX_RECORD . . . . .	??
KEY . . . . .	??
KEYLIST . . . . .	??
OPEN_RECORD . . . . .	??
PROGRAM_INFO . . . . .	??
RECORD_LIST . . . . .	??
REQUEST_LIST . . . . .	??
RUNINFO . . . . .	??
TAG . . . . .	??

## Chapter 4

# Midas File Index

### 4.1 Midas File List

Here is a list of all files with brief descriptions:

<b>analyzer.dox</b>	.. . . . .	??
<b>appendixA.dox</b>	.. . . . .	??
<b>appendixB.dox</b>	.. . . . .	??
<b>appendixC.dox</b>	.. . . . .	??
<b>appendixD.dox</b>	.. . . . .	??
<b>appendixE.dox</b>	.. . . . .	??
<b>appendixG.dox</b>	.. . . . .	??
<b>components.dox</b>	.. . . . .	??
<b>ebuser.c</b>	.. . . . .	??
<b>esone.c</b>	.. . . . .	??
<b>eventbuilder.dox</b>	.. . . . .	??
<b>internal.dox</b>	.. . . . .	??
<b>introduction.dox</b>	.. . . . .	??
<b>mcstd.h</b>	.. . . . .	??
<b>mevb.c</b>	.. . . . .	??
<b>mfe.c</b>	.. . . . .	??
<b>mhttpd.dox</b>	.. . . . .	??
<b>midas.c</b>	.. . . . .	??
<b>midas.dox</b>	.. . . . .	??
<b>midas.h</b>	.. . . . .	??
<b>mrpc.c</b>	.. . . . .	??
<b>mrpc.h</b>	.. . . . .	??
<b>msystem.h</b>	.. . . . .	??
<b>mvmestd.h</b>	.. . . . .	??
<b>newdocfeatures.dox</b>	.. . . . .	??

odb.c . . . . .	??
odbstruct.dox . . . . .	??
quickstart.dox . . . . .	??
system.c . . . . .	??
utilities.dox . . . . .	??
ybos.c . . . . .	??
ybos.h . . . . .	??



# Chapter 5

## Midas Page Index

### 5.1 Midas Related Pages

Here is a list of all related documentation pages:

MIDAS Analyzer . . . . .	??
Data format . . . . .	??
Supported hardware . . . . .	??
CAMAC and VME access function call . . . . .	??
Midas build options and operation considerations . . . . .	??
Midas Code and Libraries . . . . .	??
Frequently Asked Questions . . . . .	??
Components . . . . .	??
Event Builder Functions . . . . .	??
Internal features . . . . .	??
Introduction . . . . .	??
mhttpd task . . . . .	??
New Documented Features . . . . .	??
ODB Structure . . . . .	??
Quick Start . . . . .	??
Utilities . . . . .	??



## Chapter 6

# Midas Module Documentation

### 6.1 Midas VME Standard

#### Modules

- **VME #define**
- **VME Functions (vme\_xxx)**

## 6.2 VME #define

### Defines

- #define **VME\_AMOD\_A32\_SB** (0x0F)
- #define **VME\_AMOD\_A32\_SP** (0x0E)
- #define **VME\_AMOD\_A32\_SD** (0x0D)
- #define **VME\_AMOD\_A32\_NB** (0x0B)
- #define **VME\_AMOD\_A32\_NP** (0x0A)
- #define **VME\_AMOD\_A32\_ND** (0x09)
- #define **VME\_AMOD\_A32\_SMBLT** (0x0C)
- #define **VME\_AMOD\_A32\_NMBLT** (0x08)
- #define **VME\_AMOD\_A24\_SB** (0x3F)
- #define **VME\_AMOD\_A24\_SP** (0x3E)
- #define **VME\_AMOD\_A24\_SD** (0x3D)
- #define **VME\_AMOD\_A24\_NB** (0x3B)
- #define **VME\_AMOD\_A24\_NP** (0x3A)
- #define **VME\_AMOD\_A24\_ND** (0x39)
- #define **VME\_AMOD\_A24\_SMBLT** (0x3C)
- #define **VME\_AMOD\_A24\_NMBLT** (0x38)
- #define **VME\_AMOD\_A16\_SD** (0x2D)
- #define **VME\_AMOD\_A16\_ND** (0x29)

### 6.2.1 Define Documentation

#### 6.2.1.1 #define VME\_A16D16 1

Definition at line 86 of file mvmestd.h.

#### 6.2.1.2 #define VME\_A16D32 2

Definition at line 87 of file mvmestd.h.

#### 6.2.1.3 #define VME\_A24D16 3

Definition at line 88 of file mvmestd.h.

#### 6.2.1.4 #define VME\_A24D32 4

Definition at line 89 of file mvmestd.h.

**6.2.1.5 `#define VME_A32D16 5`**

Definition at line 90 of file `mvmestd.h`.

**6.2.1.6 `#define VME_A32D32 6`**

Definition at line 91 of file `mvmestd.h`.

**6.2.1.7 `#define VME_AMOD_A16 BT_AMOD_A16_SD`**

Definition at line 128 of file `mvmestd.h`.

**6.2.1.8 `#define VME_AMOD_A16_ND (0x29)`**

A16 Short Non-Privileged Data Access

Definition at line 126 of file `mvmestd.h`.

**6.2.1.9 `#define VME_AMOD_A16_SD (0x2D)`**

A16 Short Supervisory Data Access

Definition at line 125 of file `mvmestd.h`.

**6.2.1.10 `#define VME_AMOD_A24 VME_AMOD_A24_SD`**

Definition at line 122 of file `mvmestd.h`.

**6.2.1.11 `#define VME_AMOD_A24_D64`  
`VME_AMOD_A24_SMBLT`**

Definition at line 123 of file `mvmestd.h`.

**6.2.1.12 `#define VME_AMOD_A24_NB (0x3B)`**

A24 Standard Non-Privileged Block Transfer

Definition at line 116 of file `mvmestd.h`.

**6.2.1.13 `#define VME_AMOD_A24_ND (0x39)`**

A24 Standard Non-Privileged Data Access

Definition at line 118 of file mvmestd.h.

**6.2.1.14    `#define VME_AMOD_A24_NMBLT (0x38)`**

A24 Multiplexed Block Transfer (D64)

Definition at line 120 of file mvmestd.h.

**6.2.1.15    `#define VME_AMOD_A24_NP (0x3A)`**

A24 Standard Non-Privileged Program Access

Definition at line 117 of file mvmestd.h.

**6.2.1.16    `#define VME_AMOD_A24_SB (0x3F)`**

A24 Standard Supervisory Block Transfer

Definition at line 113 of file mvmestd.h.

**6.2.1.17    `#define VME_AMOD_A24_SD (0x3D)`**

A24 Standard Supervisory Data Access

Definition at line 115 of file mvmestd.h.

**6.2.1.18    `#define VME_AMOD_A24_SMBLT (0x3C)`**

A24 Multiplexed Block Transfer (D64)

Definition at line 119 of file mvmestd.h.

**6.2.1.19    `#define VME_AMOD_A24_SP (0x3E)`**

A24 Standard Supervisory Program Access

Definition at line 114 of file mvmestd.h.

**6.2.1.20    `#define VME_AMOD_A32 VME_AMOD_A32_SD`**

Definition at line 110 of file mvmestd.h.

**6.2.1.21** `#define VME_AMOD_A32_D64`  
`VME_AMOD_A32_SMBLT`

Definition at line 111 of file `vmestd.h`.

**6.2.1.22** `#define VME_AMOD_A32_NB (0x0B)`

A32 Extended Non-Privileged Block

Definition at line 104 of file `vmestd.h`.

**6.2.1.23** `#define VME_AMOD_A32_ND (0x09)`

A32 Extended Non-Privileged Data

Definition at line 106 of file `vmestd.h`.

**6.2.1.24** `#define VME_AMOD_A32_NMBLT (0x08)`

A32 Multiplexed Block Transfer (D64)

Definition at line 108 of file `vmestd.h`.

**6.2.1.25** `#define VME_AMOD_A32_NP (0x0A)`

A32 Extended Non-Privileged Program

Definition at line 105 of file `vmestd.h`.

**6.2.1.26** `#define VME_AMOD_A32_SB (0x0F)`

A32 Extended Supervisory Block

Definition at line 101 of file `vmestd.h`.

**6.2.1.27** `#define VME_AMOD_A32_SD (0x0D)`

A32 Extended Supervisory Data

Definition at line 103 of file `vmestd.h`.

**6.2.1.28** `#define VME_AMOD_A32_SMBLT (0x0C)`

A32 Multiplexed Block Transfer (D64)

Definition at line 107 of file mvmestd.h.

**6.2.1.29    `#define VME_AMOD_A32_SP (0x0E)`**

A32 Extended Supervisory Program

Definition at line 102 of file mvmestd.h.

**6.2.1.30    `#define VME_IOCTL_AMOD_GET 1`**

Definition at line 98 of file mvmestd.h.

**6.2.1.31    `#define VME_IOCTL_AMOD_SET 0`**

Definition at line 97 of file mvmestd.h.

**6.2.1.32    `#define VME_LM 9`**

Definition at line 94 of file mvmestd.h.

**6.2.1.33    `#define VME_RAMD16 7`**

Definition at line 92 of file mvmestd.h.

**6.2.1.34    `#define VME_RAND32 8`**

Definition at line 93 of file mvmestd.h.



## 6.3 VME Functions (`vme_`xxx)

### Functions

- int EXPRT **vme\_open** (int device, int mode)
- int EXPRT **vme\_ioctl** (int *vh*, int req, int \*parm)
- int EXPRT **vme\_close** (int *vh*)
- int EXPRT **vme\_read** (int *vh*, void \*dst, int vme\_addr, int am, int size, int dma)
- int EXPRT **vme\_write** (int *vh*, void \*src, int vme\_addr, int am, int size, int dma)
- int EXPRT **vme\_mmap** (int *vh*, void \*\*ptr, int vme\_addr, int size)
- int EXPRT **vme\_unmap** (int *vh*, void \*ptr, int size)

### 6.3.1 Function Documentation

#### 6.3.1.1 int EXPRT vme\_close (int *vh*)

**Parameters:**

*vh* ...

**Returns:**

SUCCESS

#### 6.3.1.2 int EXPRT vme\_ioctl (int *vh*, int *req*, int \* *parm*)

**Parameters:**

*vh* ...

*req* ...

*parm* ...

**Returns:**

SUCCESS

#### 6.3.1.3 int EXPRT vme\_mmap (int *vh*, void \*\* *ptr*, int *vme\_addr*, int *size*)

**Parameters:**

*vh* ...

*ptr* ...

*vme\_addr*  
*size*

**Returns:**

SUCCESS

**6.3.1.4** int EXPRT vme\_open (int *device*, int *mode*)

**Parameters:**

*device* ....  
*mode* ...

**Returns:**

SUCCESS

**6.3.1.5** int EXPRT vme\_read (int *vh*, void \* *dst*, int *vme\_addr*, int *am*, int *size*, int *dma*)

**Parameters:**

*vh* ...  
*dst* ...  
*vme\_addr*  
*am*  
*size*  
*dma*

**Returns:**

SUCCESS

**6.3.1.6** int EXPRT vme\_unmap (int *vh*, void \* *ptr*, int *size*)

**Parameters:**

*vh* ....  
*ptr* ...  
*size*

**Returns:**

SUCCESS

6.3.1.7 int EXPRT *vme\_write* (int *vh*, void \* *src*, int *vme\_addr*,  
int *am*, int *size*, int *dma*)

**Parameters:**

*vh* ...  
*src* ...  
*vme\_addr*  
*am*  
*size*  
*dma*

**Returns:**

SUCCESS

## 6.4 Midas CAMAC standard

### Modules

- Camac Functions (camxxx)

## 6.5 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*)

### 6.5.1 Function Documentation

#### 6.5.1.1 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

#### 6.5.1.2 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 response (0:no Q, 1: Q)

**Returns:**

void

Referenced by csmad(), and cssa().

### 6.5.1.3 **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

### 6.5.1.4 **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

### 6.5.1.5 **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

**6.5.1.6 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

**6.5.1.7 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

**6.5.1.8 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 response (0:no Q, 1: Q)

**Returns:**

void

Referenced by `cssa()`.

**6.5.1.9 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 repetition

**Returns:**

void

**6.5.1.10** **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

**6.5.1.11** **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().

**6.5.1.12 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

**6.5.1.13 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

**6.5.1.14 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

**6.5.1.15 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_sn(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

**6.5.1.16** **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

**6.5.1.17** **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().

**6.5.1.18** **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 repetition

**Returns:**

void

**6.5.1.19** **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 response (0:no Q, 1: Q)

**Returns:**

void

**6.5.1.20** **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

**6.5.1.21** **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



**6.5.1.22 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

**6.5.1.23 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

**6.5.1.24** **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 response (0:no Q, 1: Q)

**Returns:**

void

**6.5.1.25** **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 repetition

**Returns:**

void

**6.5.1.26** **EXTERNAL INLINE** void **EXPRT** cam\_crate\_clear  
(const int *c*)

Issue CLEAR to crate.

**Parameters:**

*c* crate number (0..)

**Returns:**

void

Referenced by cccc().

**6.5.1.27 EXTERNAL INLINE void EXPRT cam\_crate\_zinit  
(const int c)**

Issue Z to crate.

**Parameters:**

*c* crate number (0..)

**Returns:**

void

Referenced by cccz().

**6.5.1.28 EXTERNAL INLINE void EXPRT cam\_exit (void)**

Close CAMAC accesss.

**6.5.1.29 EXTERNAL INLINE void EXPRT cam\_inhibit\_clear  
(const int c)**

Clear Crate inhibit.

**Parameters:**

*c* crate number (0..)

**Returns:**

void

Referenced by ccci().

**6.5.1.30 EXTERNAL INLINE void EXPRT cam\_inhibit\_set  
(const int c)**

Set Crate inhibit.

**Parameters:***c* crate number (0..)**Returns:**

void

Referenced by `ccci()`.**6.5.1.31 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()`.**6.5.1.32 EXTERNAL INLINE int EXPRT cam\_init (void)**

Initialize CAMAC access.

**Returns:**

1: success

Referenced by `ccinit()`, and `fcinit()`.**6.5.1.33 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 ~~Parameters~~ **Parameters** only.*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

**6.5.1.34 EXTERNAL INLINE void EXPRT cam\_interrupt\_attach**  
**(const int *c*, const int *n*, void(\* *isr*)(void))**

Attach service routine to LAM of specific crate and station.

**Parameters:**

*c* crate number (0..)

*n* station number

(\**isr*) Function pointer to attach to the LAM

**Returns:**

void

Referenced by cclnk().

**6.5.1.35 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().

**6.5.1.36 EXTERNAL INLINE void EXPRT**  
**cam\_interrupt\_disable (const int *c*)**

Disables interrupts in specific crate

**Parameters:**

*c* crate number (0..)

**Returns:**

void

Referenced by cccd().

**6.5.1.37   EXTERNAL INLINE void EXPRT cam\_interrupt\_enable  
(const int *c*)**

Enable interrupts in specific crate

**Parameters:**

*c* crate number (0..)

**Returns:**

void

Referenced by cccd(), and ccrgl().

**6.5.1.38   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().

**6.5.1.39   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(), and ccrgl().

**6.5.1.40 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

**6.5.1.41 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(), and ccrgl().

**6.5.1.42 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().





**6.5.1.46** **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(), and ctm().

**6.5.1.47** **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

**6.5.1.48** **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

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

Same as **cam16i()**(p. ??)

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

Same as **cam16o()**(p. ??)

## 6.6 The midas.h & midas.c

### Modules

- Midas #define
- Midas Macros
- Midas Error definition
- Midas Structure Declaration
- Midas Message Functions (msg\_XXX)
- Midas Common Functions (cm\_XXX)
- Midas Buffer Manager Functions (bm\_XXX)
- Midas RPC Functions (rpc\_XXX)
- Midas Bank Functions (bk\_XXX)
- Midas History Functions (hs\_XXX)
- Midas Elog Functions (el\_XXX)
- Midas Alarm Functions (al\_XXX)
- Midas Dual Buffer Memory Functions (dm\_XXX)

### 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 32
- #define MAX\_EVENT\_REQUESTS 10
- #define MAX\_OPEN\_RECORDS 100
- #define MAX\_ODB\_PATH 256
- #define MAX\_EXPERIMENT 32
- #define BANKLIST\_MAX 32
- #define STRING\_BANKLIST\_MAX BANKLIST\_MAX \* 4
- #define CH\_BS 8
- #define LAM\_SOURCE(c, s) (c < 24 | ((s) & 0xFFFFFFFF))
- #define LAM\_STATION(s) (1 < (s-1))
- #define LAM\_SOURCE\_CRATE(c) (c > 24)

- `#define LAM_SOURCE_STATION(s) ((s) & 0xFFFFF)`
- `#define CNAF 0x1`
- `#define ANA_CONTINUE 1`

### 6.6.1 Define Documentation

#### 6.6.1.1 `#define ANA_CONTINUE 1`

dox\*\*\*\*\*

Definition at line 1063 of file midas.h.

#### 6.6.1.2 `#define ANA_SKIP 0`

Definition at line 1064 of file midas.h.

#### 6.6.1.3 `#define BANKLIST_MAX 32`

max # of banks in event

Definition at line 578 of file midas.h.

Referenced by `bk_list()`.

#### 6.6.1.4 `#define CH_BS 8`

special characters

Definition at line 723 of file midas.h.

#### 6.6.1.5 `#define CH_CR 13`

Definition at line 725 of file midas.h.

#### 6.6.1.6 `#define CH_DELETE (CH_EXT+2)`

Definition at line 731 of file midas.h.

#### 6.6.1.7 `#define CH_DOWN (CH_EXT+7)`

Definition at line 736 of file midas.h.

**6.6.1.8 #define CH\_END (CH\_EXT+3)**

Definition at line 732 of file midas.h.

**6.6.1.9 #define CH\_EXT 0x100**

Definition at line 727 of file midas.h.

**6.6.1.10 #define CH\_HOME (CH\_EXT+0)**

Definition at line 729 of file midas.h.

**6.6.1.11 #define CH\_INSERT (CH\_EXT+1)**

Definition at line 730 of file midas.h.

**6.6.1.12 #define CH\_LEFT (CH\_EXT+9)**

Definition at line 738 of file midas.h.

**6.6.1.13 #define CH\_PDOWN (CH\_EXT+5)**

Definition at line 734 of file midas.h.

**6.6.1.14 #define CH\_PUP (CH\_EXT+4)**

Definition at line 733 of file midas.h.

**6.6.1.15 #define CH\_RIGHT (CH\_EXT+8)**

Definition at line 737 of file midas.h.

**6.6.1.16 #define CH\_TAB 9**

Definition at line 724 of file midas.h.

**6.6.1.17 #define CH\_UP (CH\_EXT+6)**

Definition at line 735 of file midas.h.

**6.6.1.18 #define CNAF 0x1**

CNAF commands

Definition at line 768 of file midas.h.

**6.6.1.19 #define CNAF\_CRATE\_CLEAR 0x102**

Definition at line 773 of file midas.h.

**6.6.1.20 #define CNAF\_CRATE\_ZINIT 0x103**

Definition at line 774 of file midas.h.

**6.6.1.21 #define CNAF\_INHIBIT\_CLEAR 0x101**

Definition at line 772 of file midas.h.

**6.6.1.22 #define CNAF\_INHIBIT\_SET 0x100**

Definition at line 771 of file midas.h.

**6.6.1.23 #define CNAF\_nQ 0x2**

Definition at line 769 of file midas.h.

**6.6.1.24 #define CNAF\_TEST 0x110**

Definition at line 775 of file midas.h.

**6.6.1.25 #define DEFAULT\_EVENT\_BUFFER\_SIZE 0x200000;**

2M

Definition at line 568 of file midas.h.

**6.6.1.26 #define DEFAULT\_ODB\_SIZE 0x100000**

online database 1M

Definition at line 569 of file midas.h.

Referenced by `cm_connect_experiment()`, `cm_connect_experiment1()`, and `main()`.

#### 6.6.1.27 `#define DEFAULT_RPC_TIMEOUT 10000`

Definition at line 585 of file `midas.h`.

#### 6.6.1.28 `#define DEFAULT_WATCHDOG_TIMEOUT 10000`

Definition at line 588 of file `midas.h`.

Referenced by `cm_connect_experiment()`, and `cm_connect_experiment1()`.

#### 6.6.1.29 `#define EVENT_BUFFER_NAME "SYSTEM"`

buffer name for commands

Definition at line 566 of file `midas.h`.

#### 6.6.1.30 `#define EVENT_BUFFER_SIZE 0x100000`

buffer used for events

Definition at line 565 of file `midas.h`.

Referenced by `main()`, and `source_booking()`.

#### 6.6.1.31 `#define HOST_NAME_LENGTH 256`

length of TCP/IP names

Definition at line 572 of file `midas.h`.

Referenced by `cm_connect_client()`, `cm_connect_experiment1()`, `cm_disconnect_experiment()`, `cm_set_client_info()`, `cm_shutdown()`, `cm_transition()`, and `main()`.

#### 6.6.1.32 `#define LAM_SOURCE(c, s) (c<<24 | ((s) & 0xFFFFF))`

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

##### Parameters:

*c* Crate number

*s* Slot number

Definition at line 746 of file midas.h.

**6.6.1.33** `#define LAM_SOURCE_CRATE(c) (c>>24)`

Convert the coded LAM crate to Crate number.

**Parameters:**

*c* coded crate

Definition at line 758 of file midas.h.

**6.6.1.34** `#define LAM_SOURCE_STATION(s) ((s) & 0xFFFFF)`

Convert the coded LAM station to Station number.

**Parameters:**

*s* Slot number

Definition at line 764 of file midas.h.

**6.6.1.35** `#define LAM_STATION(s) (1<<(s-1))`

Code the Station number bitwise for the LAM source.

**Parameters:**

*s* Slot number

Definition at line 752 of file midas.h.

**6.6.1.36** `#define MAX_CLIENTS 32`

client processes per buf/db

Definition at line 573 of file midas.h.

Referenced by `bm_close_buffer()`, `bm_open_buffer()`, `cm_cleanup()`, `db_close_database()`, and `db_open_database()`.

**6.6.1.37** `#define MAX_EVENT_REQUESTS 10`

event requests per client

Definition at line 574 of file midas.h.

Referenced by `bm_remove_event_request()`.



**6.6.1.38 #define MAX\_EVENT\_SIZE 0x80000**

maximum event size 512k

Definition at line 567 of file midas.h.

Referenced by `bm_send_event()`, `dm_buffer_create()`, `main()`, and `rpc_send_event()`.

**6.6.1.39 #define MAX\_EXPERIMENT 32**

number of different exp.

Definition at line 577 of file midas.h.

Referenced by `cm_connect_experiment1()`, `cm_list_experiments()`, `cm_scan_experiments()`, and `cm_select_experiment()`.

**6.6.1.40 #define MAX\_ODB\_PATH 256**

length of path in ODB

Definition at line 576 of file midas.h.

Referenced by `db_copy()`.

**6.6.1.41 #define MAX\_OPEN\_RECORDS 100**

number of open DB records

Definition at line 575 of file midas.h.

**6.6.1.42 #define MIDAS\_TCP\_PORT 1175**

Definition at line 582 of file midas.h.

Referenced by `cm_list_experiments()`, and `cm_transition()`.

**6.6.1.43 #define MIDAS\_VERSION "1.9.3"**

Definition at line 418 of file midas.h.

Referenced by `cm_get_version()`.

**6.6.1.44 #define NAME\_LENGTH 32**

length of names, mult.of 8!

Definition at line 571 of file midas.h.

Referenced by `bm_open_buffer()`, `cm_check_client()`, `cm_connect_client()`, `cm_connect_experiment1()`, `cm_exist()`, `cm_get_client_info()`, `cm_list_experiments()`, `cm_select_experiment()`, `cm_set_client_info()`, `cm_shutdown()`, `cm_transition()`, and `db_open_database()`.

#### **6.6.1.45 `#define NET_TCP_SIZE 0xFFFF`**

maximum TCP transfer

Definition at line 561 of file midas.h.

Referenced by `rpc_send_event()`, and `scheduler()`.

#### **6.6.1.46 `#define NET_UDP_SIZE 8192`**

maximum UDP transfer

Definition at line 563 of file midas.h.

#### **6.6.1.47 `#define OPT_TCP_SIZE 8192`**

optimal TCP buffer size

Definition at line 562 of file midas.h.

#### **6.6.1.48 `#define STRING_BANKLIST_MAX BANKLIST_MAX * 4`**

for `bk_list()`(p.??)

Definition at line 579 of file midas.h.

#### **6.6.1.49 `#define TAPE_BUFFER_SIZE 0x8000`**

buffer size for taping data

Definition at line 559 of file midas.h.

#### **6.6.1.50 `#define WATCHDOG_INTERVAL 1000`**

Definition at line 586 of file midas.h.

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

## 6.6.2 Variable Documentation

### 6.6.2.1 `INT _call_watchdog = TRUE` [static]

Definition at line 1658 of file midas.c.

Referenced by `cm_get_watchdog_params()`, and `cm_set_watchdog_params()`.

### 6.6.2.2 `char _client_name[NAME_LENGTH]` [static]

Definition at line 1656 of file midas.c.

### 6.6.2.3 `HNDLE _hDB = 0` [static]

Definition at line 1655 of file midas.c.

Referenced by `cm_get_experiment_database()`, and `cm_set_experiment_database()`.

### 6.6.2.4 `HNDLE _hKeyClient = 0` [static]

Definition at line 1654 of file midas.c.

Referenced by `cm_connect_experiment1()`, `cm_get_experiment_database()`, and `cm_set_experiment_database()`.

### 6.6.2.5 `INT _mutex_alarm`

Definition at line 1660 of file midas.c.

Referenced by `db_close_database()`.

### 6.6.2.6 `INT _mutex_eolog`

Definition at line 1660 of file midas.c.

Referenced by `db_close_database()`.

### 6.6.2.7 `char _path_name[MAX_STRING_LENGTH]` [static]

Definition at line 1657 of file midas.c.

Referenced by `cm_get_path()`, and `cm_set_path()`.

#### 6.6.2.8 INT \_watchdog\_timeout = DEFAULT\_WATCHDOG\_TIMEOUT [static]

Definition at line 1659 of file midas.c.

Referenced by cm\_get\_watchdog\_params(), and cm\_set\_watchdog\_params().

## 6.7 Midas #define

### Defines

- #define **STATE\_STOPPED** 1
- #define **STATE\_PAUSED** 2
- #define **STATE\_RUNNING** 3
- #define **FORMAT\_MIDAS** 1
- #define **FORMAT\_YBOS** 2
- #define **FORMAT\_ASCII** 3
- #define **FORMAT\_FIXED** 4
- #define **FORMAT\_DUMP** 5
- #define **FORMAT\_HBOOK** 6
- #define **FORMAT\_ROOT** 7
- #define **GET\_ALL** (1<<0)
- #define **GET\_SOME** (1<<1)
- #define **GET\_FARM** (1<<2)
- #define **TID\_BYTE** 1
- #define **TID\_SBYTE** 2
- #define **TID\_CHAR** 3
- #define **TID\_WORD** 4
- #define **TID\_SHORT** 5
- #define **TID\_DWORD** 6
- #define **TID\_INT** 7
- #define **TID\_BOOL** 8
- #define **TID\_FLOAT** 9
- #define **TID\_DOUBLE** 10
- #define **TID\_BITFIELD** 11
- #define **TID\_STRING** 12
- #define **TID\_ARRAY** 13
- #define **TID\_STRUCT** 14
- #define **TID\_KEY** 15
- #define **TID\_LINK** 16
- #define **TID\_LAST** 17
- #define **SYNC** 0
- #define **MODE\_READ** (1<<0)
- #define **RPC\_OTIMEOUT** 1
- #define **WF\_WATCH\_ME** (1<<0)
- #define **TR\_START** (1<<0)
- #define **TR\_STOP** (1<<1)
- #define **TR\_PAUSE** (1<<2)
- #define **TR\_RESUME** (1<<3)
- #define **EQ\_PERIODIC** (1<<0)

- #define **EQ\_POLLED** (1<<1)
- #define **EQ\_INTERRUPT** (1<<2)
- #define **EQ\_SLOW** (1<<3)
- #define **EQ\_MANUAL\_TRIG** (1<<4)
- #define **EQ\_FRAGMENTED** (1<<5)
- #define **RO\_RUNNING** (1<<0)
- #define **RO\_STOPPED** (1<<1)
- #define **RO\_PAUSED** (1<<2)
- #define **RO\_BOR** (1<<3)
- #define **RO\_EOR** (1<<4)
- #define **RO\_PAUSE** (1<<5)
- #define **RO\_RESUME** (1<<6)
- #define **RO\_TRANSITIONS** (RO\_BOR|RO\_EOR|RO\_PAUSE|RO\_RESUME)
- #define **RO\_ALWAYS** (0xFF)
- #define **RO\_ODB** (1<<8)
- #define **MT\_ERROR** (1<<0)
- #define **MT\_INFO** (1<<1)
- #define **MT\_DEBUG** (1<<2)
- #define **MT\_USER** (1<<3)
- #define **MT\_LOG** (1<<4)
- #define **MT\_TALK** (1<<5)
- #define **MT\_CALL** (1<<6)
- #define **MT\_ALL** 0xFF
- #define **MERROR** MT\_ERROR, \_\_FILE\_\_, \_\_LINE\_\_
- #define **MINFO** MT\_INFO, \_\_FILE\_\_, \_\_LINE\_\_
- #define **MDEBUG** MT\_DEBUG, \_\_FILE\_\_, \_\_LINE\_\_
- #define **MUSER** MT\_USER, \_\_FILE\_\_, \_\_LINE\_\_
- #define **MLOG** MT\_LOG, \_\_FILE\_\_, \_\_LINE\_\_
- #define **MTALK** MT\_TALK, \_\_FILE\_\_, \_\_LINE\_\_
- #define **MCALL** MT\_CALL, \_\_FILE\_\_, \_\_LINE\_\_

## 6.7.1 Define Documentation

### 6.7.1.1 #define ASYNC 1

Definition at line 645 of file midas.h.

Referenced by `bm_receive_event()`, `cm_transition()`, `handFlush()`, `main()`, `rpc_send_event()`, `scheduler()`, and `source_scan()`.

**6.7.1.2 #define EQ\_FRAGMENTED (1<<5)**

Fragmented Event

Definition at line 701 of file midas.h.

Referenced by send\_event().

**6.7.1.3 #define EQ\_INTERRUPT (1<<2)**

Interrupt Event

Definition at line 698 of file midas.h.

Referenced by scheduler().

**6.7.1.4 #define EQ\_MANUAL\_TRIG (1<<4)**

Manual triggered Event

Definition at line 700 of file midas.h.

**6.7.1.5 #define EQ\_PERIODIC (1<<0)**

Periodic Event

Definition at line 696 of file midas.h.

Referenced by scheduler().

**6.7.1.6 #define EQ\_POLLED (1<<1)**

Polling Event

Definition at line 697 of file midas.h.

Referenced by scheduler().

**6.7.1.7 #define EQ\_SLOW (1<<3)**

Slow Control Event

Definition at line 699 of file midas.h.

Referenced by main(), scheduler(), and send\_event().

**6.7.1.8 #define EVENTID\_ALL -1**

Definition at line 814 of file midas.h.

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

**6.7.1.9 #define FORMAT\_ASCII 3**

ASCII format

Definition at line 609 of file midas.h.

**6.7.1.10 #define FORMAT\_DUMP 5**

Dump (detailed ASCII) format

Definition at line 611 of file midas.h.

**6.7.1.11 #define FORMAT\_FIXED 4**

Fixed length binary records

Definition at line 610 of file midas.h.

Referenced by `update_odb()`.

**6.7.1.12 #define FORMAT\_HBOOK 6**

CERN hbook (rz) format

Definition at line 612 of file midas.h.

**6.7.1.13 #define FORMAT\_MIDAS 1**

MIDAS banks

Definition at line 607 of file midas.h.

Referenced by `main()`, `source_scan()`, and `update_odb()`.

**6.7.1.14 #define FORMAT\_ROOT 7**

CERN ROOT format

Definition at line 613 of file midas.h.



**6.7.1.15 `#define FORMAT_YBOS 2`**

YBOS banks

Definition at line 608 of file midas.h.

Referenced by `main()`, `source_scan()`, and `update_odb()`.

**6.7.1.16 `#define GET_ALL (1<<0)`**

get all events (consume)

Definition at line 617 of file midas.h.

Referenced by `bm_flush_cache()`, `bm_remove_event_request()`, `bm_send_event()`, and `source_booking()`.

**6.7.1.17 `#define GET_FARM (1<<2)`**

distribute events over several clients (farming)

Definition at line 619 of file midas.h.

**6.7.1.18 `#define GET_SOME (1<<1)`**

get as much as possible (sampling)

Definition at line 618 of file midas.h.

Referenced by `cm_msg_register()`.

**6.7.1.19 `#define MCALL MT_CALL, __FILE__, __LINE__`**

info message for telephone call

Definition at line 834 of file midas.h.

**6.7.1.20 `#define MDEBUG MT_DEBUG, __FILE__,`  
`__LINE__`**

•

Definition at line 830 of file midas.h.

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

**6.7.1.21** `#define MERROR MT_ERROR, __FILE__,  
__LINE__`

•

Definition at line 828 of file midas.h.

Referenced by `al_trigger_alarm()`, `bm_close_buffer()`, `bm_flush_cache()`, `bm_open_buffer()`, `bm_push_event()`, `bm_receive_event()`, `bm_remove_event_request()`, `bm_request_event()`, `bm_send_event()`, `bm_set_cache_size()`, `bm_skip_event()`, `cm_check_client()`, `cm_check_deferred_transition()`, `cm_cleanup()`, `cm_connect_experiment1()`, `cm_get_watchdog_info()`, `cm_list_experiments()`, `cm_register_deferred_transition()`, `cm_set_client_info()`, `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_find_key()`, `db_get_data()`, `db_get_data_index()`, `db_get_key()`, `db_get_key_info()`, `db_get_key_time()`, `db_get_record()`, `db_get_value()`, `db_load()`, `db_lock_database()`, `db_open_database()`, `db_open_record()`, `db_paste()`, `db_protect_database()`, `db_save()`, `db_save_struct()`, `db_set_data()`, `db_set_data_index()`, `db_set_record()`, `db_set_value()`, `db_unlock_database()`, `dm_buffer_create()`, `el_submit()`, `handFlush()`, `interrupt_routine()`, `main()`, `rpc_flush_event()`, `rpc_register_functions()`, `rpc_send_event()`, `rpc_set_option()`, `scheduler()`, `send_event()`, `source_booking()`, `source_scan()`, `source_unbooking()`, `tr_prestart()`, and `update_odb()`.

**6.7.1.22** `#define MINFO MT_INFO, __FILE__, __LINE__`

•

Definition at line 829 of file midas.h.

Referenced by `bk_list()`, `cm_check_client()`, `cm_cleanup()`, `cm_connect_experiment1()`, `cm_disconnect_experiment()`, `cm_set_client_info()`, `cm_shutdown()`, `cm_transition()`, `main()`, and `ybk_list()`.

**6.7.1.23** `#define MLOG MT_LOG, __FILE__, __LINE__`

info message which is only logged

Definition at line 832 of file midas.h.

**6.7.1.24** `#define MODE_ALLOC (1<<7)`

Definition at line 653 of file midas.h.

Referenced by `db_close_all_records()`, `db_close_record()`, and `db_open_record()`.

#### 6.7.1.25 `#define MODE_DELETE (1<<2)`

Definition at line 651 of file `midas.h`.

Referenced by `cm_delete_client_info()`, `cm_set_client_info()`, `cm_transition()`, `db_create_key()`, `db_delete_key1()`, and `db_open_database()`.

#### 6.7.1.26 `#define MODE_EXCLUSIVE (1<<3)`

Definition at line 652 of file `midas.h`.

Referenced by `cm_cleanup()`, `db_create_key()`, `db_open_record()`, `db_set_data()`, `db_set_data_index()`, and `db_set_value()`.

#### 6.7.1.27 `#define MODE_READ (1<<0)`

Access modes

Definition at line 649 of file `midas.h`.

Referenced by `cm_delete_client_info()`, `cm_register_deferred_transition()`, `cm_register_transition()`, `cm_set_client_info()`, `cm_set_watchdog_params()`, `cm_transition()`, `db_create_key()`, `db_find_key()`, `db_get_data()`, `db_get_data_index()`, `db_get_value()`, `db_open_database()`, and `db_open_record()`.

#### 6.7.1.28 `#define MODE_WRITE (1<<1)`

Definition at line 650 of file `midas.h`.

Referenced by `cm_cleanup()`, `cm_delete_client_info()`, `cm_register_deferred_transition()`, `cm_register_transition()`, `cm_set_client_info()`, `cm_set_watchdog_params()`, `cm_transition()`, `db_close_all_records()`, `db_close_record()`, `db_create_key()`, `db_open_database()`, `db_open_record()`, `db_send_changed_records()`, `db_set_data()`, `db_set_data_index()`, `db_set_value()`, and `db_update_record()`.

#### 6.7.1.29 `#define MT_ALL 0xFF`

•

Definition at line 826 of file midas.h.

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

#### **6.7.1.30   `#define MT_CALL (1<<6)`**

- 

Definition at line 825 of file midas.h.

#### **6.7.1.31   `#define MT_DEBUG (1<<2)`**

- 

Definition at line 821 of file midas.h.

Referenced by `cm_msg_log()`, and `cm_msg_log1()`.

#### **6.7.1.32   `#define MT_ERROR (1<<0)`**

- 

Definition at line 819 of file midas.h.

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

#### **6.7.1.33   `#define MT_INFO (1<<1)`**

- 

Definition at line 820 of file midas.h.

#### **6.7.1.34   `#define MT_LOG (1<<4)`**

- 

Definition at line 823 of file midas.h.

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

#### **6.7.1.35   `#define MT_TALK (1<<5)`**

- 

Definition at line 824 of file midas.h.

**6.7.1.36 `#define MT_USER (1<<3)`**

- 

Definition at line 822 of file midas.h.

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

**6.7.1.37 `#define MTALK MT_TALK, __FILE__, __LINE__`**

info message for speech system

Definition at line 833 of file midas.h.

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

**6.7.1.38 `#define MUSER MT_USER, __FILE__, __LINE__`**

produced by interactive user

Definition at line 831 of file midas.h.

**6.7.1.39 `#define RO_ALWAYS (0xFF)`**

Always (independent of the run status)

Definition at line 714 of file midas.h.

**6.7.1.40 `#define RO_BOR (1<<3)`**

At the Begin of run

Definition at line 708 of file midas.h.

Referenced by `send_all_periodic_events()`.

**6.7.1.41 `#define RO_EOR (1<<4)`**

At the End of run

Definition at line 709 of file midas.h.

Referenced by `send_all_periodic_events()`.

**6.7.1.42 `#define RO_ODB (1<<8)`**

Submit data to ODB only

Definition at line 716 of file midas.h.

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

#### **6.7.1.43    `#define RO_PAUSE (1<<5)`**

Before pausing the run

Definition at line 710 of file midas.h.

Referenced by `send_all_periodic_events()`.

#### **6.7.1.44    `#define RO_PAUSED (1<<2)`**

???

Definition at line 707 of file midas.h.

Referenced by `scheduler()`.

#### **6.7.1.45    `#define RO_RESUME (1<<6)`**

Before resuming the run

Definition at line 711 of file midas.h.

Referenced by `send_all_periodic_events()`.

#### **6.7.1.46    `#define RO_RUNNING (1<<0)`**

While running

Definition at line 705 of file midas.h.

Referenced by `scheduler()`.

#### **6.7.1.47    `#define RO_STOPPED (1<<1)`**

Before stopping the run

Definition at line 706 of file midas.h.

Referenced by `scheduler()`.

#### **6.7.1.48    `#define RO_TRANSITIONS (RO_BOR|RO_EOR|RO_PAUSE|RO_RESUME)`**

At all transitions

Definition at line 713 of file midas.h.

#### **6.7.1.49 `#define RPC_CLIENT_HANDLE 9`**

Definition at line 665 of file midas.h.

Referenced by `cm_get_experiment_database()`, and `cm_set_client_info()`.

#### **6.7.1.50 `#define RPC_CONVERT_FLAGS 7`**

Definition at line 663 of file midas.h.

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

#### **6.7.1.51 `#define RPC_FTCP 1`**

Definition at line 671 of file midas.h.

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

#### **6.7.1.52 `#define RPC_NODELAY 12`**

Definition at line 668 of file midas.h.

Referenced by `rpc_set_option()`.

#### **6.7.1.53 `#define RPC_OCONVERT_FLAG 3`**

Definition at line 659 of file midas.h.

#### **6.7.1.54 `#define RPC_ODB_HANDLE 8`**

Definition at line 664 of file midas.h.

Referenced by `cm_get_experiment_database()`, and `cm_set_client_info()`.

#### **6.7.1.55 `#define RPC_OHW_TYPE 4`**

Definition at line 660 of file midas.h.

Referenced by `cm_connect_experiment1()`.

**6.7.1.56 #define RPC\_OSERVER\_NAME 6**

Definition at line 662 of file midas.h.

**6.7.1.57 #define RPC\_OSERVER\_TYPE 5**

Definition at line 661 of file midas.h.

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

**6.7.1.58 #define RPC\_OTIMEOUT 1**

RPC options

Definition at line 657 of file midas.h.

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

**6.7.1.59 #define RPC\_OTRANSPORT 2**

Definition at line 658 of file midas.h.

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

**6.7.1.60 #define RPC\_SEND SOCK 10**

Definition at line 666 of file midas.h.

**6.7.1.61 #define RPC\_TCP 0**

Definition at line 670 of file midas.h.

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

**6.7.1.62 #define RPC\_WATCHDOG\_TIMEOUT 11**

Definition at line 667 of file midas.h.

Referenced by `cm_set_watchdog_params()`.



**6.7.1.63 #define STATE\_PAUSED 2**

MIDAS run paused

Definition at line 602 of file midas.h.

Referenced by cm\_transition(), and scheduler().

**6.7.1.64 #define STATE\_RUNNING 3**

MIDAS run running

Definition at line 603 of file midas.h.

Referenced by cm\_transition(), display(), main(), scheduler(), and tr\_prestart().

**6.7.1.65 #define STATE\_STOPPED 1**

MIDAS run stopped

Definition at line 601 of file midas.h.

Referenced by cm\_transition(), display(), handFlush(), main(), and scheduler().

**6.7.1.66 #define SYNC 0**

Synchronous / Asynchronous flags

Definition at line 644 of file midas.h.

Referenced by cm\_check\_deferred\_transition(), cm\_msg(), cm\_msg1(), interrupt\_routine(), main(), scheduler(), send\_event(), and source\_scan().

**6.7.1.67 #define TID\_ARRAY 13**

array with unknown contents

Definition at line 636 of file midas.h.

**6.7.1.68 #define TID\_BITFIELD 11**

32 Bits Bitfield 0 111... (32)

Definition at line 634 of file midas.h.

Referenced by db\_sprintf().

**6.7.1.69 #define TID\_BOOL 8**

four bytes bool 0 1

Definition at line 631 of file midas.h.

Referenced by al\_trigger\_alarm(), bk\_swap(), db\_sprintf(), and scheduler().

**6.7.1.70 #define TID\_BYTE 1**

unsigned byte 0 255

Definition at line 624 of file midas.h.

Referenced by db\_sprintf().

**6.7.1.71 #define TID\_CHAR 3**

single character 0 255

Definition at line 626 of file midas.h.

Referenced by db\_sprintf().

**6.7.1.72 #define TID\_DOUBLE 10**

8 Byte float format

Definition at line 633 of file midas.h.

Referenced by bk\_swap(), and db\_sprintf().

**6.7.1.73 #define TID\_DWORD 6**

four bytes 0 2<sup>32</sup>-1

Definition at line 629 of file midas.h.

Referenced by bk\_swap(), bm\_receive\_event(), cm\_register\_deferred\_transition(), cm\_register\_transition(), cm\_set\_client\_info(), cm\_transition(), db\_sprintf(), and db\_update\_record().

**6.7.1.74 #define TID\_FLOAT 9**

4 Byte float format

Definition at line 632 of file midas.h.

Referenced by bk\_swap(), and db\_sprintf().

**6.7.1.75 `#define TID_INT 7`**

signed dword  $-2^{31}$   $2^{31}-1$

Definition at line 630 of file midas.h.

Referenced by `al_trigger_alarm()`, `bk_swap()`, `cm_connect_client()`, `cm_connect_experiment1()`, `cm_delete_client_info()`, `cm_register_deferred_transition()`, `cm_set_client_info()`, `cm_set_watchdog_params()`, `cm_shutdown()`, `cm_transition()`, `db_sprintf()`, `el_submit()`, `main()`, and `scheduler()`.

**6.7.1.76 `#define TID_KEY 15`**

key in online database

Definition at line 638 of file midas.h.

Referenced by `cm_transition()`, `db_check_record()`, `db_copy()`, `db_create_key()`, `db_create_record()`, `db_delete_key1()`, `db_enum_key()`, `db_find_key()`, `db_get_data()`, `db_get_data_index()`, `db_get_key_info()`, `db_get_record()`, `db_get_record_size()`, `db_open_database()`, `db_paste()`, `db_set_data()`, `db_set_data_index()`, `db_set_record()`, `db_set_value()`, and `main()`.

**6.7.1.77 `#define TID_LAST 17`**

end of TID list indicator

Definition at line 640 of file midas.h.

Referenced by `db_check_record()`, `db_create_key()`, and `db_paste()`.

**6.7.1.78 `#define TID_LINK 16`**

link in online database

Definition at line 639 of file midas.h.

Referenced by `db_check_record()`, `db_copy()`, `db_create_key()`, `db_create_link()`, `db_delete_key1()`, `db_enum_key()`, `db_find_key()`, `db_get_value()`, `db_paste()`, `db_set_data_index()`, `db_set_value()`, `db_sprintf()`, and `update_odb()`.

**6.7.1.79 `#define TID_SBYTE 2`**

signed byte  $-128$   $127$

Definition at line 625 of file midas.h.

Referenced by db\_sprintf().

#### **6.7.1.80 #define TID\_SHORT 5**

signed word -32768 32767

Definition at line 628 of file midas.h.

Referenced by bk\_swap(), bm\_receive\_event(), and db\_sprintf().

#### **6.7.1.81 #define TID\_STRING 12**

zero terminated string

Definition at line 635 of file midas.h.

Referenced by al\_trigger\_alarm(), 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\_set\_data\_index(), db\_set\_value(), db\_sprintf(), el\_submit(), logger\_root(), main(), tr\_prestart(), and update\_odb().

#### **6.7.1.82 #define TID\_STRUCT 14**

structure with fixed length

Definition at line 637 of file midas.h.

Referenced by bk\_close(), and update\_odb().

#### **6.7.1.83 #define TID\_WORD 4**

two bytes 0 65535

Definition at line 627 of file midas.h.

Referenced by bk\_swap(), and db\_sprintf().

#### **6.7.1.84 #define TR\_DEFERRED (1<<12)**

Definition at line 692 of file midas.h.

Referenced by cm\_check\_deferred\_transition(), and cm\_transition().

**6.7.1.85 `#define TR_PAUSE (1<<2)`**

Pause transition

Definition at line 682 of file midas.h.

Referenced by `cm_transition()`, and `send_all_periodic_events()`.

**6.7.1.86 `#define TR_POSTPAUSE (1<<9)`**

Definition at line 689 of file midas.h.

Referenced by `cm_transition()`.

**6.7.1.87 `#define TR_POSTRESUME (1<<11)`**

Definition at line 691 of file midas.h.

Referenced by `cm_transition()`.

**6.7.1.88 `#define TR_POSTSTART (1<<5)`**

Definition at line 685 of file midas.h.

Referenced by `cm_transition()`.

**6.7.1.89 `#define TR_POSTSTOP (1<<7)`**

Definition at line 687 of file midas.h.

Referenced by `cm_transition()`.

**6.7.1.90 `#define TR_PREPAUSE (1<<8)`**

Definition at line 688 of file midas.h.

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

**6.7.1.91 `#define TR_PRERESUME (1<<10)`**

Definition at line 690 of file midas.h.

Referenced by `cm_transition()`.

**6.7.1.92 #define TR\_PRESTART (1<<4)**

Definition at line 684 of file midas.h.

Referenced by cm\_transition(), and main().

**6.7.1.93 #define TR\_PRESTOP (1<<6)**

Definition at line 686 of file midas.h.

Referenced by cm\_transition(), and main().

**6.7.1.94 #define TR\_RESUME (1<<3)**

Resume transition

Definition at line 683 of file midas.h.

Referenced by cm\_transition(), main(), and send\_all\_periodic\_events().

**6.7.1.95 #define TR\_START (1<<0)**

Start transition

Definition at line 680 of file midas.h.

Referenced by cm\_transition(), main(), scheduler(), and send\_all\_periodic\_events().

**6.7.1.96 #define TR\_STOP (1<<1)**

Stop transition

Definition at line 681 of file midas.h.

Referenced by cm\_transition(), main(), scheduler(), and send\_all\_periodic\_events().

**6.7.1.97 #define TRIGGER\_ALL -1**

Definition at line 815 of file midas.h.

Referenced by bm\_match\_event(), and cm\_msg\_register().

**6.7.1.98 #define WF\_CALL\_WD (1<<1)**

Definition at line 676 of file midas.h.

**6.7.1.99** `#define WF_WATCH_ME (1<<0)`

Watchdog flags

Definition at line 675 of file midas.h.

## 6.8 Midas Macros

### Defines

- `#define max(a, b) (((a) > (b)) ? (a) : (b))`
- `#define min(a, b) (((a) < (b)) ? (a) : (b))`
- `#define ALIGN(x) (((x)+7) & ~7)`
- `#define VALIGN(adr, align) (((PTYPE) (adr)+align-1) & ~(align-1))`

### 6.8.1 Define Documentation

#### 6.8.1.1 `#define ALIGN(x) (((x)+7) & ~7)`

Align macro for data alignment on 8-byte boundary

Definition at line 799 of file midas.h.

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

#### 6.8.1.2 `#define max(a, b) (((a) > (b)) ? (a) : (b))`

`max`

Definition at line 786 of file midas.h.

Referenced by `cm_execute()`, and `scheduler()`.

#### 6.8.1.3 `#define min(a, b) (((a) < (b)) ? (a) : (b))`

`min`

Definition at line 792 of file midas.h.

Referenced by `update_odb()`.

#### 6.8.1.4 `#define VALIGN(adr, align) (((PTYPE) (adr)+align-1) & ~(align-1))`

Align macro for variable data alignment

Definition at line 803 of file midas.h.

Referenced by `db_get_record_size()`, and `update_odb()`.



## 6.9 Midas Error definition

### Modules

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

## 6.10 Midas Structure Declaration

### Modules

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

## 6.11 Status and error codes

### Defines

- `#define SUCCESS 1`
- `#define CM_SUCCESS 1`
- `#define CM_SET_ERROR 102`
- `#define CM_NO_CLIENT 103`
- `#define CM_DB_ERROR 104`
- `#define CM_UNDEF_EXP 105`
- `#define CM_VERSION_MISMATCH 106`
- `#define CM_SHUTDOWN 107`
- `#define CM_WRONG_PASSWORD 108`
- `#define CM_UNDEF_ENVIRON 109`
- `#define CM_DEFERRED_TRANSITION 110`
- `#define CM_TRANSITION_IN_PROGRESS 111`

#### 6.11.1 Define Documentation

##### 6.11.1.1 `#define CM_DB_ERROR 104`

db access error

Definition at line 853 of file midas.h.

Referenced by `cm_msg_retrieve()`.

##### 6.11.1.2 `#define CM_DEFERRED_TRANSITION 110`

•

Definition at line 859 of file midas.h.

Referenced by `cm_transition()`.

##### 6.11.1.3 `#define CM_NO_CLIENT 103`

nobody

Definition at line 852 of file midas.h.

Referenced by `cm_check_client()`, `cm_connect_client()`, `cm_exist()`, `cm_get_watchdog_info()`, `cm_set_client_info()`, and `cm_shutdown()`.

**6.11.1.4 #define CM\_SET\_ERROR 102**

set

Definition at line 851 of file midas.h.

**6.11.1.5 #define CM\_SHUTDOWN 107**

•

Definition at line 856 of file midas.h.

**6.11.1.6 #define CM\_SUCCESS 1**

Same

Definition at line 850 of file midas.h.

Referenced by bk\_swap(), cm\_asctime(), cm\_check\_client(), cm\_check\_deferred\_transition(), cm\_cleanup(), cm\_connect\_experiment(), cm\_connect\_experiment1(), cm\_delete\_client\_info(), cm\_disconnect\_experiment(), cm\_execute(), cm\_exist(), cm\_get\_client\_info(), cm\_get\_environment(), cm\_get\_error(), cm\_get\_experiment\_database(), cm\_get\_path(), cm\_get\_watchdog\_info(), cm\_get\_watchdog\_params(), cm\_list\_experiments(), cm\_msg(), cm\_msg1(), cm\_msg\_log(), cm\_msg\_log1(), cm\_msg\_retrieve(), cm\_register\_deferred\_transition(), cm\_register\_transition(), cm\_scan\_experiments(), cm\_select\_experiment(), cm\_set\_client\_info(), cm\_set\_experiment\_database(), cm\_set\_path(), cm\_set\_watchdog\_params(), cm\_shutdown(), cm\_synchronize(), cm\_time(), cm\_transition(), dm\_buffer\_create(), eb\_mfragment\_add(), eb\_yfragment\_add(), main(), scheduler(), send\_event(), tr\_prestart(), and tr\_stop().

**6.11.1.7 #define CM\_TRANSITION\_IN\_PROGRESS 111**

•

Definition at line 860 of file midas.h.

Referenced by cm\_transition().

**6.11.1.8 #define CM\_UNDEF\_ENVIRON 109**

•

Definition at line 858 of file midas.h.

Referenced by `cm_scan_experiments()`.

#### 6.11.1.9 `#define CM_UNDEF_EXP 105`

- 

Definition at line 854 of file `midas.h`.

Referenced by `cm_connect_experiment1()`, and `cm_get_client_info()`.

#### 6.11.1.10 `#define CM_VERSION_MISMATCH 106`

- 

Definition at line 855 of file `midas.h`.

#### 6.11.1.11 `#define CM_WRONG_PASSWORD 108`

- 

Definition at line 857 of file `midas.h`.

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

#### 6.11.1.12 `#define SUCCESS 1`

Success

Definition at line 849 of file `midas.h`.

Referenced by `cm_check_deferred_transition()`, `cm_transition()`, `el_submit()`, `fccinit()`, `main()`, `scheduler()`, `source_booking()`, and `tr_prestart()`.

## 6.12 Buffer Manager error codes

### Defines

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

### 6.12.1 Define Documentation

#### 6.12.1.1 `#define BM_ASYNC_RETURN 209`

- 

Definition at line 876 of file `midas.h`.

Referenced by `bm_flush_cache()`, `bm_receive_event()`, `bm_send_event()`, `main()`, `rpc_send_event()`, and `source_scan()`.

#### 6.12.1.2 `#define BM_CONFLICT 213`

- 

Definition at line 880 of file `midas.h`.

Referenced by `source_booking()`.

**6.12.1.3 #define BM\_CREATED 202**

- 

Definition at line 869 of file midas.h.

Referenced by `bm_open_buffer()`, `cm_msg()`, `cm_msg1()`, `cm_msg_register()`, `main()`, and `source_booking()`.

**6.12.1.4 #define BM\_EXIT 214**

- 

Definition at line 881 of file midas.h.

**6.12.1.5 #define BM\_INVALID\_HANDLE 205**

- 

Definition at line 872 of file midas.h.

Referenced by `bm_close_buffer()`, `bm_delete_request()`, `bm_flush_cache()`, `bm_push_event()`, `bm_receive_event()`, `bm_remove_event_request()`, `bm_send_event()`, `bm_set_cache_size()`, and `bm_skip_event()`.

**6.12.1.6 #define BM\_INVALID\_MIXING 217**

- 

Definition at line 884 of file midas.h.

**6.12.1.7 #define BM\_INVALID\_NAME 204**

- 

Definition at line 871 of file midas.h.

Referenced by `bm_open_buffer()`.

**6.12.1.8 #define BM\_INVALID\_PARAM 215**

-

Definition at line 882 of file midas.h.

Referenced by `bm_open_buffer()`, `bm_send_event()`, `bm_set_cache_size()`, and `rpc_send_event()`.

#### **6.12.1.9    `#define BM_MEMSIZE_MISMATCH 212`**

•

Definition at line 879 of file midas.h.

Referenced by `bm_open_buffer()`, and `dm_buffer_create()`.

#### **6.12.1.10   `#define BM_MORE_EVENTS 216`**

•

Definition at line 883 of file midas.h.

Referenced by `bm_check_buffers()`, and `bm_push_event()`.

#### **6.12.1.11   `#define BM_MULTIPLE_HOSTS 211`**

•

Definition at line 878 of file midas.h.

#### **6.12.1.12   `#define BM_NO_MEMORY 203`**

•

Definition at line 870 of file midas.h.

Referenced by `bm_open_buffer()`, `bm_push_event()`, `bm_request_event()`, `bm_send_event()`, `bm_set_cache_size()`, `dm_buffer_create()`, and `source_booking()`.

#### **6.12.1.13   `#define BM_NO_MUTEX 207`**

•

Definition at line 874 of file midas.h.

Referenced by `bm_open_buffer()`.



**6.12.1.14 #define BM\_NO\_SHM 218**

- 

Definition at line 885 of file midas.h.

Referenced by `bm_open_buffer()`.

**6.12.1.15 #define BM\_NO\_SLOT 206**

- 

Definition at line 873 of file midas.h.

Referenced by `bm_open_buffer()`.

**6.12.1.16 #define BM\_NOT\_FOUND 208**

- 

Definition at line 875 of file midas.h.

Referenced by `bm_remove_event_request()`.

**6.12.1.17 #define BM\_SUCCESS 1**

- 

Definition at line 868 of file midas.h.

Referenced by `bm_close_all_buffers()`, `bm_close_buffer()`, `bm_compose_event()`, `bm_delete_request()`, `bm_empty_buffers()`, `bm_flush_cache()`, `bm_open_buffer()`, `bm_push_event()`, `bm_receive_event()`, `bm_remove_event_request()`, `bm_request_event()`, `bm_send_event()`, `bm_set_cache_size()`, `bm_skip_event()`, `cm_msg()`, `cm_msg1()`, `cm_msg_register()`, `cm_set_msg_print()`, `handFlush()`, `main()`, `scheduler()`, `send_event()`, `source_booking()`, `source_scan()`, and `source_unbooking()`.

**6.12.1.18 #define BM\_TRUNCATED 210**

- 

Definition at line 877 of file midas.h.

Referenced by `bm_receive_event()`.

## 6.13 Online Database error codes

### Defines

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

### 6.13.1 Define Documentation

#### 6.13.1.1 `#define DB_CORRUPTED 323`

- 

Definition at line 914 of file `midas.h`.

Referenced by `db_create_key()`, and `db_find_key()`.

**6.13.1.2 #define DB\_CREATED 302**

- 

Definition at line 893 of file midas.h.

Referenced by cm\_connect\_experiment1(), and db\_open\_database().

**6.13.1.3 #define DB\_FILE\_ERROR 317**

- 

Definition at line 908 of file midas.h.

Referenced by db\_load(), db\_save(), and db\_save\_struct().

**6.13.1.4 #define DB\_FULL 310**

- 

Definition at line 901 of file midas.h.

Referenced by db\_create\_key(), db\_set\_data(), db\_set\_data\_index(), and db\_set\_value().

**6.13.1.5 #define DB\_INVALID\_HANDLE 305**

- 

Definition at line 896 of file midas.h.

Referenced by cm\_get\_watchdog\_info(), db\_close\_database(), db\_close\_record(), db\_create\_key(), db\_delete\_key1(), db\_enum\_key(), db\_find\_key(), db\_get\_data(), db\_get\_data\_index(), db\_get\_key(), db\_get\_key\_info(), db\_get\_key\_time(), db\_get\_value(), db\_lock\_database(), db\_protect\_database(), db\_save\_struct(), db\_set\_data(), db\_set\_data\_index(), db\_unlock\_database(), and db\_update\_record().

**6.13.1.6 #define DB\_INVALID\_LINK 322**

- 

Definition at line 913 of file midas.h.

Referenced by db\_find\_key().

**6.13.1.7 #define DB\_INVALID\_NAME 304**

- 

Definition at line 895 of file midas.h.

Referenced by db\_open\_database().

**6.13.1.8 #define DB\_INVALID\_PARAM 309**

- 

Definition at line 900 of file midas.h.

Referenced by db\_create\_key(), db\_open\_database(), db\_set\_data(), and db\_set\_value().

**6.13.1.9 #define DB\_KEY\_CREATED 313**

- 

Definition at line 904 of file midas.h.

**6.13.1.10 #define DB\_KEY\_EXIST 311**

- 

Definition at line 902 of file midas.h.

Referenced by db\_create\_key().

**6.13.1.11 #define DB\_MEMSIZE\_MISMATCH 308**

- 

Definition at line 899 of file midas.h.

**6.13.1.12 #define DB\_NO\_ACCESS 318**

-

Definition at line 909 of file midas.h.

Referenced by db\_create\_key(), db\_delete\_key1(), db\_find\_key(), db\_get\_data(), db\_get\_data\_index(), db\_get\_value(), db\_open\_record(), db\_set\_data(), db\_set\_data\_index(), and db\_set\_value().

#### **6.13.1.13 #define DB\_NO\_KEY 312**

- 

Definition at line 903 of file midas.h.

Referenced by cm\_exist(), cm\_shutdown(), db\_check\_record(), db\_create\_key(), db\_create\_link(), db\_create\_record(), db\_find\_key(), db\_get\_value(), db\_paste(), and db\_set\_value().

#### **6.13.1.14 #define DB\_NO\_MEMORY 303**

- 

Definition at line 894 of file midas.h.

Referenced by db\_copy(), db\_load(), db\_open\_database(), db\_open\_record(), and db\_paste().

#### **6.13.1.15 #define DB\_NO\_MORE\_SUBKEYS 316**

- 

Definition at line 907 of file midas.h.

Referenced by cm\_connect\_client(), cm\_exist(), cm\_set\_client\_info(), cm\_shutdown(), cm\_transition(), db\_enum\_key(), logger\_root(), and update\_odb().

#### **6.13.1.16 #define DB\_NO\_MUTEX 307**

- 

Definition at line 898 of file midas.h.

Referenced by db\_open\_database().

**6.13.1.17 #define DB\_NO\_SLOT 306**

- 

Definition at line 897 of file midas.h.

Referenced by db\_open\_database().

**6.13.1.18 #define DB\_OPEN\_RECORD 320**

- 

Definition at line 911 of file midas.h.

Referenced by db\_create\_record(), and db\_delete\_key1().

**6.13.1.19 #define DB\_OUT\_OF\_RANGE 321**

- 

Definition at line 912 of file midas.h.

Referenced by db\_get\_data\_index().

**6.13.1.20 #define DB\_STRUCT\_MISMATCH 324**

- 

Definition at line 915 of file midas.h.

Referenced by cm\_connect\_experiment1(), and db\_check\_record().

**6.13.1.21 #define DB\_STRUCT\_SIZE\_MISMATCH 319**

- 

Definition at line 910 of file midas.h.

Referenced by db\_get\_record(), db\_open\_record(), and db\_set\_record().

**6.13.1.22 #define DB\_SUCCESS 1**

-

Definition at line 892 of file midas.h.

Referenced by `al_trigger_alarm()`, `cm_connect_client()`, `cm_connect_experiment1()`, `cm_delete_client_info()`, `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_close_all_records()`, `db_close_database()`, `db_close_record()`, `db_copy()`, `db_create_key()`, `db_create_link()`, `db_create_record()`, `db_delete_key1()`, `db_enum_key()`, `db_find_key()`, `db_get_data()`, `db_get_data_index()`, `db_get_key()`, `db_get_key_info()`, `db_get_key_time()`, `db_get_record()`, `db_get_record_size()`, `db_get_value()`, `db_lock_database()`, `db_open_database()`, `db_open_record()`, `db_paste()`, `db_protect_database()`, `db_save()`, `db_save_struct()`, `db_send_changed_records()`, `db_set_data()`, `db_set_data_index()`, `db_set_record()`, `db_set_value()`, `db_sprintf()`, `db_unlock_database()`, `db_update_record()`, `el_submit()`, `logger_root()`, `main()`, `scheduler()`, and `update_odb()`.

#### 6.13.1.23 `#define DB_TRUNCATED 314`

•

Definition at line 905 of file midas.h.

Referenced by `db_check_record()`, `db_copy()`, `db_create_record()`, `db_get_data()`, `db_get_data_index()`, `db_get_value()`, `db_paste()`, and `db_save()`.

#### 6.13.1.24 `#define DB_TYPE_MISMATCH 315`

•

Definition at line 906 of file midas.h.

Referenced by `db_get_data()`, `db_get_data_index()`, `db_get_value()`, `db_set_data()`, `db_set_data_index()`, and `db_set_value()`.

## 6.14 System Services error code

### Defines

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

### 6.14.1 Define Documentation

#### 6.14.1.1 `#define SS_ABORT 415`

- 

Definition at line 936 of file `midas.h`.

Referenced by `bm_flush_cache()`, `bm_receive_event()`, `bm_send_event()`, `main()`, and `scheduler()`.



**6.14.1.2 #define SS\_CLIENT\_RECV 414**

- 

Definition at line 935 of file midas.h.

**6.14.1.3 #define SS\_CREATED 402**

- 

Definition at line 923 of file midas.h.

Referenced by `bm_open_buffer()`, `cm_connect_experiment1()`, `db_open_database()`, and `dm_buffer_create()`.

**6.14.1.4 #define SS\_DEV\_BUSY 418**

- 

Definition at line 939 of file midas.h.

**6.14.1.5 #define SS\_END\_OF\_FILE 423**

- 

Definition at line 944 of file midas.h.

**6.14.1.6 #define SS\_END\_OF\_TAPE 422**

- 

Definition at line 943 of file midas.h.

**6.14.1.7 #define SS\_EXIT 416**

- 

Definition at line 937 of file midas.h.

**6.14.1.8 #define SS\_FILE\_ERROR 407**

- 

Definition at line 928 of file midas.h.

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

**6.14.1.9 #define SS\_FILE\_EXISTS 424**

- 

Definition at line 945 of file midas.h.

**6.14.1.10 #define SS\_INVALID\_ADDRESS 406**

- 

Definition at line 927 of file midas.h.

**6.14.1.11 #define SS\_INVALID\_FORMAT 426**

- 

Definition at line 947 of file midas.h.

**6.14.1.12 #define SS\_INVALID\_HANDLE 405**

- 

Definition at line 926 of file midas.h.

**6.14.1.13 #define SS\_INVALID\_NAME 404**

- 

Definition at line 925 of file midas.h.

**6.14.1.14 #define SS\_IO\_ERROR 419**

- 

Definition at line 940 of file midas.h.

**6.14.1.15** `#define SS_NO_DRIVER 421`

- 

Definition at line 942 of file midas.h.

**6.14.1.16** `#define SS_NO_MEMORY 403`

- 

Definition at line 924 of file midas.h.

Referenced by `bm_open_buffer()`, `db_open_database()`, `scheduler()`, and `send_event()`.

**6.14.1.17** `#define SS_NO_MUTEX 408`

- 

Definition at line 929 of file midas.h.

**6.14.1.18** `#define SS_NO_PROCESS 409`

- 

Definition at line 930 of file midas.h.

**6.14.1.19** `#define SS_NO_ROOT 427`

- 

Definition at line 948 of file midas.h.

**6.14.1.20** `#define SS_NO_SPACE 425`

- 

Definition at line 946 of file midas.h.

**6.14.1.21 #define SS\_NO\_TAPE 417**

- 

Definition at line 938 of file midas.h.

**6.14.1.22 #define SS\_NO\_THREAD 410**

- 

Definition at line 931 of file midas.h.

Referenced by `ss_thread_kill()`.

**6.14.1.23 #define SS\_SERVER\_RECV 413**

- 

Definition at line 934 of file midas.h.

**6.14.1.24 #define SS\_SOCKET\_ERROR 411**

- 

Definition at line 932 of file midas.h.

Referenced by `ss_sleep()`.

**6.14.1.25 #define SS\_SUCCESS 1**

- 

Definition at line 922 of file midas.h.

Referenced by `bm_open_buffer()`, `bm_receive_event()`, `cm_connect_experiment1()`, `db_open_database()`, `dm_buffer_create()`, `source_scan()`, `ss_sleep()`, and `ss_thread_kill()`.

**6.14.1.26 #define SS\_TAPE\_ERROR 420**

- 

Definition at line 941 of file midas.h.

6.14.1.27 `#define SS_TIMEOUT 412`

- 

Definition at line 933 of file midas.h.

## 6.15 Remote Procedure Calls error codes

### Defines

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

### 6.15.1 Define Documentation

#### 6.15.1.1 `#define RPC_ABORT SS_ABORT`

- 

Definition at line 956 of file midas.h.

#### 6.15.1.2 `#define RPC_CONNCLOSED 507`

- 

Definition at line 962 of file midas.h.

#### 6.15.1.3 `#define RPC_DOUBLE_DEFINED 511`

- 

Definition at line 966 of file midas.h.

Referenced by `rpc_register_functions()`.

**6.15.1.4 #define RPC\_EXCEED\_BUFFER 505**

- 

Definition at line 960 of file midas.h.

Referenced by `rpc_send_event()`.

**6.15.1.5 #define RPC\_INVALID\_ID 508**

- 

Definition at line 963 of file midas.h.

**6.15.1.6 #define RPC\_NET\_ERROR 503**

- 

Definition at line 958 of file midas.h.

Referenced by `bm_receive_event()`, `cm_connect_experiment1()`, `cm_list_experiments()`, `rpc_flush_event()`, and `rpc_send_event()`.

**6.15.1.7 #define RPC\_NO\_CONNECTION 502**

- 

Definition at line 957 of file midas.h.

**6.15.1.8 #define RPC\_NO\_MEMORY 510**

- 

Definition at line 965 of file midas.h.

Referenced by `rpc_register_functions()`.

**6.15.1.9 #define RPC\_NOT\_REGISTERED 506**

- 

Definition at line 961 of file midas.h.

**6.15.1.10 #define RPC\_SHUTDOWN 509**

- 

Definition at line 964 of file midas.h.

Referenced by `cm_yield()`, `main()`, and `scheduler()`.

**6.15.1.11 #define RPC\_SUCCESS 1**

- 

Definition at line 955 of file midas.h.

Referenced by `cm_connect_experiment1()`, `cm_shutdown()`, `cm_transition()`, `rpc_flush_event()`, `rpc_register_client()`, `rpc_register_functions()`, `rpc_send_event()`, and `scheduler()`.

**6.15.1.12 #define RPC\_TIMEOUT 504**

- 

Definition at line 959 of file midas.h.



## 6.16 Other errors

### Defines

- `#define FE_SUCCESS 1`
- `#define FE_ERR_ODB 602`
- `#define FE_ERR_HW 603`
- `#define FE_ERR_DISABLED 604`
- `#define FE_ERR_DRIVER 605`
- `#define HS_SUCCESS 1`
- `#define HS_FILE_ERROR 702`
- `#define HS_NO_MEMORY 703`
- `#define HS_TRUNCATED 704`
- `#define HS_WRONG_INDEX 705`
- `#define HS_UNDEFINED_EVENT 706`
- `#define HS_UNDEFINED_VAR 707`
- `#define FTP_SUCCESS 1`
- `#define FTP_NET_ERROR 802`
- `#define FTP_FILE_ERROR 803`
- `#define FTP_RESPONSE_ERROR 804`
- `#define FTP_INVALID_ARG 805`
- `#define EL_SUCCESS 1`
- `#define EL_FILE_ERROR 902`
- `#define EL_NO_MESSAGE 903`
- `#define EL_TRUNCATED 904`
- `#define EL_FIRST_MSG 905`
- `#define EL_LAST_MSG 906`
- `#define AL_SUCCESS 1`
- `#define AL_INVALID_NAME 1002`
- `#define AL_ERROR_ODB 1003`
- `#define AL_RESET 1004`
- `#define CMD_INIT (1<<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)`

### 6.16.1 Define Documentation

#### 6.16.1.1 `#define AL_ERROR_ODB 1003`

- 

Definition at line 1010 of file midas.h.

Referenced by `al_trigger_alarm()`.

#### 6.16.1.2 `#define AL_INVALID_NAME 1002`

- 

Definition at line 1009 of file midas.h.

#### 6.16.1.3 `#define AL_RESET 1004`

- 

Definition at line 1011 of file midas.h.

#### 6.16.1.4 `#define AL_SUCCESS 1`

- 

Definition at line 1008 of file midas.h.

Referenced by `al_trigger_alarm()`.

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

macros for bus driver access

Definition at line 1051 of file midas.h.

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

Definition at line 1053 of file midas.h.

**6.16.1.7** `#define BD_READS(s, z, p, t) info → bd(CMD_READ,  
info → bd_info, s, z, p, t)`

Definition at line 1052 of file midas.h.

**6.16.1.8** `#define BD_WRITES(s) info → bd(CMD_WRITE, info  
→ bd_info, s)`

Definition at line 1054 of file midas.h.

**6.16.1.9** `#define CMD_DEBUG 104`

Definition at line 1039 of file midas.h.

**6.16.1.10** `#define CMD_DISABLE_COMMAND (1<<16)`

Definition at line 1031 of file midas.h.

**6.16.1.11** `#define CMD_ENABLE_COMMAND (1<<15)`

Definition at line 1030 of file midas.h.

**6.16.1.12** `#define CMD_EXIT (1<<1)`

Definition at line 1016 of file midas.h.

Referenced by main().

**6.16.1.13** `#define CMD_GET (1<<5)`

Definition at line 1020 of file midas.h.

**6.16.1.14** `#define CMD_GET_ALL (1<<6)`

Definition at line 1021 of file midas.h.

**6.16.1.15** `#define CMD_GET_CURRENT (1<<7)`

Definition at line 1022 of file midas.h.

**6.16.1.16 #define CMD\_GET\_CURRENT\_ALL (1<<8)**

Definition at line 1023 of file midas.h.

**6.16.1.17 #define CMD\_GET\_DEFAULT\_NAME (1<<12)**

Definition at line 1027 of file midas.h.

**6.16.1.18 #define CMD\_GET\_DEFAULT\_THRESHOLD  
(1<<13)**

Definition at line 1028 of file midas.h.

**6.16.1.19 #define CMD\_GET\_DEMAND (1<<11)**

Definition at line 1026 of file midas.h.

**6.16.1.20 #define CMD\_GETS 103**

Definition at line 1038 of file midas.h.

**6.16.1.21 #define CMD\_IDLE (1<<2)**

Definition at line 1017 of file midas.h.

Referenced by scheduler().

**6.16.1.22 #define CMD\_INIT (1<<0)**

Slow control commands error code

Definition at line 1015 of file midas.h.

**6.16.1.23 #define CMD\_INTERRUPT\_ATTACH 102**

Definition at line 1046 of file midas.h.

**6.16.1.24 #define CMD\_INTERRUPT\_DETACH 103**

Definition at line 1047 of file midas.h.

Referenced by main().

**6.16.1.25 #define CMD\_INTERRUPT\_DISABLE 101**

Definition at line 1045 of file midas.h.

Referenced by interrupt\_enable(), and main().

**6.16.1.26 #define CMD\_INTERRUPT\_ENABLE 100**

Commands for interrupt events error code

Definition at line 1044 of file midas.h.

Referenced by interrupt\_enable().

**6.16.1.27 #define CMD\_NAME 105**

Definition at line 1040 of file midas.h.

**6.16.1.28 #define CMD\_PUTS 102**

Definition at line 1037 of file midas.h.

**6.16.1.29 #define CMD\_READ 101**

Definition at line 1036 of file midas.h.

**6.16.1.30 #define CMD\_SET (1<<3)**

Definition at line 1018 of file midas.h.

**6.16.1.31 #define CMD\_SET\_ALL (1<<4)**

Definition at line 1019 of file midas.h.

**6.16.1.32 #define CMD\_SET\_CURRENT\_LIMIT (1<<9)**

Definition at line 1024 of file midas.h.

**6.16.1.33 #define CMD\_SET\_CURRENT\_LIMIT\_ALL (1<<10)**

Definition at line 1025 of file midas.h.

**6.16.1.34 #define CMD\_SET\_LABEL (1<<14)**

Definition at line 1029 of file midas.h.

**6.16.1.35 #define CMD\_WRITE 100**

Bus driver commands

Definition at line 1035 of file midas.h.

**6.16.1.36 #define EL\_FILE\_ERROR 902**

•

Definition at line 1000 of file midas.h.

Referenced by el\_submit().

**6.16.1.37 #define EL\_FIRST\_MSG 905**

•

Definition at line 1003 of file midas.h.

**6.16.1.38 #define EL\_LAST\_MSG 906**

•

Definition at line 1004 of file midas.h.

**6.16.1.39 #define EL\_NO\_MESSAGE 903**

•

Definition at line 1001 of file midas.h.

**6.16.1.40 #define EL\_SUCCESS 1**

•

Definition at line 999 of file midas.h.

Referenced by el\_submit().

**6.16.1.41** `#define EL_TRUNCATED 904`

- 

Definition at line 1002 of file midas.h.

**6.16.1.42** `#define FE_ERR_DISABLED 604`

- 

Definition at line 976 of file midas.h.

Referenced by `display()`.

**6.16.1.43** `#define FE_ERR_DRIVER 605`

- 

Definition at line 977 of file midas.h.

**6.16.1.44** `#define FE_ERR_HW 603`

- 

Definition at line 975 of file midas.h.

Referenced by `display()`.

**6.16.1.45** `#define FE_ERR_ODB 602`

- 

Definition at line 974 of file midas.h.

Referenced by `display()`.

**6.16.1.46** `#define FE_SUCCESS 1`

- 

Definition at line 973 of file midas.h.

Referenced by `display()`, `main()`, `scheduler()`, and `send_all_periodic_events()`.

**6.16.1.47 #define FTP\_FILE\_ERROR 803**

- 

Definition at line 993 of file midas.h.

**6.16.1.48 #define FTP\_INVALID\_ARG 805**

- 

Definition at line 995 of file midas.h.

**6.16.1.49 #define FTP\_NET\_ERROR 802**

- 

Definition at line 992 of file midas.h.

**6.16.1.50 #define FTP\_RESPONSE\_ERROR 804**

- 

Definition at line 994 of file midas.h.

**6.16.1.51 #define FTP\_SUCCESS 1**

- 

Definition at line 991 of file midas.h.

**6.16.1.52 #define HS\_FILE\_ERROR 702**

- 

Definition at line 982 of file midas.h.

**6.16.1.53 #define HS\_NO\_MEMORY 703**

- 

Definition at line 983 of file midas.h.



**6.16.1.54 #define HS\_SUCCESS 1**

- 

Definition at line 981 of file midas.h.

Referenced by `hs_open_file()`, and `hs_set_path()`.

**6.16.1.55 #define HS\_TRUNCATED 704**

- 

Definition at line 984 of file midas.h.

**6.16.1.56 #define HS\_UNDEFINED\_EVENT 706**

- 

Definition at line 986 of file midas.h.

**6.16.1.57 #define HS\_UNDEFINED\_VAR 707**

- 

Definition at line 987 of file midas.h.

**6.16.1.58 #define HS\_WRONG\_INDEX 705**

- 

Definition at line 985 of file midas.h.

## 6.17 Buffer Section

### Data Structures

- struct **BUFFER**
- struct **BUFFER\_CLIENT**
- struct **BUFFER\_HEADER**
- struct **EVENT\_HEADER**
- struct **EVENT\_REQUEST**
- struct **KEY**
- struct **KEYLIST**

### Defines

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

#### 6.17.1 Define Documentation

**6.17.1.1** `#define EVENT_ID(e) (((EVENT_HEADER *) e)-1) → event_id)`

**EVENT\_ID** Extract or set the event ID field pointed by the argument..

**Parameters:**

*e* pointer to the midas event (pevent)

Definition at line 1099 of file midas.h.

**6.17.1.2** `#define EVENT_SOURCE(e, o) (* (INT*) (e+o))`

Definition at line 1114 of file midas.h.

**6.17.1.3 #define EVENTID\_BOR ((short int) 0x8000)**

Begin-of-run

Definition at line 1118 of file midas.h.

**6.17.1.4 #define EVENTID\_EOR ((short int) 0x8001)**

End-of-run

Definition at line 1119 of file midas.h.

**6.17.1.5 #define EVENTID\_FRAG ((unsigned short) 0xD000)**

Definition at line 1125 of file midas.h.

Referenced by `bm_match_event()`, `bm_push_event()`, and `send_event()`.

**6.17.1.6 #define EVENTID\_FRAG1 ((unsigned short) 0xC000)**

fragmented events

Definition at line 1124 of file midas.h.

Referenced by `bm_match_event()`, `bm_push_event()`, and `send_event()`.

**6.17.1.7 #define EVENTID\_MESSAGE ((short int) 0x8002)**

Message events

Definition at line 1120 of file midas.h.

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

**6.17.1.8 #define MIDAS\_MAGIC 0x494d**

'MI'

Definition at line 1129 of file midas.h.

**6.17.1.9 #define SERIAL\_NUMBER(e) (((EVENT\_HEADER \*) e)-1) → serial\_number)**

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

**Parameters:**

*e* pointer to the midas event (pevent)

Definition at line 1106 of file midas.h.

**6.17.1.10** `#define TIME_STAMP(e) (((EVENT_HEADER *)  
e)-1) → time_stamp)`

TIME\_STAMP Extract or set/reset the time stamp field pointed by the argument.

**Parameters:**

*e* pointer to the midas event (pevent)

Definition at line 1113 of file midas.h.

**6.17.1.11** `#define TRIGGER_MASK(e) (((EVENT_HEADER  
*) e)-1) → trigger_mask)`

TRIGGER\_MASK Extract or set the trigger mask field pointed by the argument.

**Parameters:**

*e* pointer to the midas event (pevent)

Definition at line 1092 of file midas.h.

## 6.18 Equipment related

### Data Structures

- struct **BUS\_DRIVER**
- struct **DEVICE\_DRIVER**
- struct **eqpmnt**
- struct **EQUIPMENT\_INFO**
- struct **EQUIPMENT\_STATS**

### Defines

- #define **DF\_INPUT** (1<<0)
- #define **DF\_OUTPUT** (1<<1)
- #define **DF\_PRIO\_DEVICE** (1<<2)
- #define **DF\_READ\_ONLY** (1<<3)

#### 6.18.1 Define Documentation

##### 6.18.1.1 #define **DF\_INPUT** (1<<0)

channel is input

Definition at line 1232 of file midas.h.

##### 6.18.1.2 #define **DF\_OUTPUT** (1<<1)

channel is output

Definition at line 1233 of file midas.h.

##### 6.18.1.3 #define **DF\_PRIO\_DEVICE** (1<<2)

get demand values from device instead of ODB

Definition at line 1234 of file midas.h.

##### 6.18.1.4 #define **DF\_READ\_ONLY** (1<<3)

never write demand values to device

Definition at line 1235 of file midas.h.

## 6.18.2 Typedef Documentation

### 6.18.2.1 typedef struct eqpmnt EQUIPMENT

Referenced by scheduler(), and send\_event().

### 6.18.2.2 typedef struct eqpmnt\* PEQUIPMENT

Definition at line 1276 of file midas.h.

## 6.19 Bank related

### Data Structures

- struct **BANK**
- struct **BANK32**
- struct **BANK\_HEADER**
- struct **BANK\_LIST**
- struct **TAG**

### Defines

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

#### 6.19.1 Define Documentation

##### 6.19.1.1 `#define BANK_FORMAT_32BIT (1<<4)`

- 

Definition at line 1311 of file midas.h.

Referenced by `bk_close()`, `bk_create()`, `bk_init32()`, and `bk_swap()`.

##### 6.19.1.2 `#define BANK_FORMAT_VERSION 1`

- 

Definition at line 1310 of file midas.h.

Referenced by `bk_init()`, and `bk_init32()`.

## 6.20 Analyzer related

### Data Structures

- struct **ANA\_MODULE**
- struct **ANA\_TEST**
- struct **ANALYZE\_REQUEST**
- struct **AR\_INFO**
- struct **AR\_STATS**

### 6.20.1 Define Documentation

**6.20.1.1** `#define DEF_TEST(t) extern ANA_TEST t;`

Definition at line 1425 of file midas.h.

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

Definition at line 1419 of file midas.h.

**6.20.1.3** `#define TEST(t) (t.value)`

Definition at line 1420 of file midas.h.



## 6.21 History related

### Data Structures

- struct **DEF\_RECORD**
- struct **HIST\_RECORD**
- struct **HISTORY**
- struct **INDEX\_RECORD**

### 6.21.1 Define Documentation

#### 6.21.1.1 `#define RT_DATA (*((DWORD *) "HSDA"))`

Definition at line 1436 of file midas.h.

#### 6.21.1.2 `#define RT_DEF (*((DWORD *) "HSDF"))`

Definition at line 1437 of file midas.h.

## 6.22 ODB runinfo related

### Data Structures

- struct **RUNINFO**

#### 6.22.1 Define Documentation

##### 6.22.1.1 #define RUNINFO\_STR(\_name)

Value:

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

Definition at line 1494 of file midas.h.

Referenced by `cm_connect_experiment1()`.

## 6.23 Alarm related

### 6.23.1 Detailed Description

Alarm structre.

#### Data Structures

- struct **ALARM**
- struct **ALARM\_CLASS**
- struct **PROGRAM\_INFO**

#### Defines

- #define **AT\_INTERNAL** 1
- #define **AT\_PROGRAM** 2
- #define **AT\_EVALUATED** 3
- #define **AT\_PERIODIC** 4
- #define **AT\_LAST** 4

### 6.23.2 Define Documentation

#### 6.23.2.1 #define ALARM\_CLASS\_STR(\_name)

Value:

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

Definition at line 1566 of file midas.h.

**6.23.2.2 #define ALARM\_ODB\_STR(\_name)****Value:**

```

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

```

Definition at line 1596 of file midas.h.

Referenced by al\_trigger\_alarm().

**6.23.2.3 #define ALARM\_PERIODIC\_STR(\_name)****Value:**

```

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

```

Definition at line 1611 of file midas.h.

**6.23.2.4 #define AT\_EVALUATED 3**

•

Definition at line 1533 of file midas.h.

Referenced by al\_trigger\_alarm().

**6.23.2.5 #define AT\_INTERNAL 1**

- 

Definition at line 1531 of file midas.h.

**6.23.2.6 #define AT\_LAST 4**

- 

Definition at line 1535 of file midas.h.

Referenced by al\_trigger\_alarm().

**6.23.2.7 #define AT\_PERIODIC 4**

- 

Definition at line 1534 of file midas.h.

Referenced by al\_trigger\_alarm().

**6.23.2.8 #define AT\_PROGRAM 2**

- 

Definition at line 1532 of file midas.h.

**6.23.2.9 #define PROGRAM\_INFO\_STR(\_name)**

Value:

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

Definition at line 1537 of file midas.h.

Referenced by `cm_set_client_info()`.

## 6.24 The ybos.h & ybos.c

### Modules

- YBOS `#define`
- YBOS error code
- YBOS Macros
- YBOS Bank Functions (`ybk_`xxx)

## 6.25 YBOS #define

### Defines

- #define YBOS\_PHYREC\_SIZE 8192
- #define YBOS\_BUFFER\_SIZE 3\*(YBOS\_PHYREC\_SIZE<<2) + MAX\_EVENT\_SIZE + 128
- #define YB\_BANKLIST\_MAX 32
- #define YB\_STRING\_BANKLIST\_MAX YB\_BANKLIST\_MAX \* 4
- #define H\_BLOCK\_SIZE 0
- #define H\_BLOCK\_NUM 1
- #define H\_HEAD\_LEN 2
- #define H\_START 3
- #define D\_RECORD 1
- #define D\_HEADER 2
- #define D\_EVTLEN 3
- #define YB\_COMPLETE 1
- #define YB\_INCOMPLETE 2
- #define YB\_NO\_RECOVER -1
- #define YB\_NO\_RUN 0
- #define YB\_ADD\_RUN 1
- #define DSP\_RAW 1
- #define DSP\_BANK 2
- #define DSP\_UNK 0
- #define DSP\_DEC 1
- #define DSP\_HEX 2
- #define DSP\_ASC 3
- #define I2\_BKTYPE 1
- #define A1\_BKTYPE 2
- #define I4\_BKTYPE 3
- #define F4\_BKTYPE 4
- #define D8\_BKTYPE 5
- #define I1\_BKTYPE 8
- #define MAX\_BKTYPE I1\_BKTYPE+1

### 6.25.1 Define Documentation

#### 6.25.1.1 #define A1\_BKTYPE 2

ASCII 1 byte

Definition at line 330 of file ybos.h.

Referenced by update\_odb().



**6.25.1.2 #define D8\_BKTYPE 5**

Double 8 bytes

Definition at line 333 of file ybos.h.

Referenced by update\_odb().

**6.25.1.3 #define D\_EVTLEN 3**

YBOS

Definition at line 163 of file ybos.h.

**6.25.1.4 #define D\_HEADER 2**

YBOS

Definition at line 162 of file ybos.h.

**6.25.1.5 #define D\_RECORD 1**

YBOS

Definition at line 161 of file ybos.h.

**6.25.1.6 #define DSP\_ASC 3**

Display data in ASCII format

Definition at line 183 of file ybos.h.

**6.25.1.7 #define DSP\_BANK 2**

Display data in bank format

Definition at line 176 of file ybos.h.

**6.25.1.8 #define DSP\_DEC 1**

Display data in decimal format

Definition at line 181 of file ybos.h.

**6.25.1.9    #define DSP\_HEX 2**

Display data in headecimal format

Definition at line 182 of file ybos.h.

**6.25.1.10   #define DSP\_RAW 1**

Display raw data

Definition at line 175 of file ybos.h.

**6.25.1.11   #define DSP\_UNK 0**

Display format unknown

Definition at line 180 of file ybos.h.

**6.25.1.12   #define F4\_BKTYPE 4**

Float 4 bytes

Definition at line 332 of file ybos.h.

**6.25.1.13   #define H\_BLOCK\_NUM 1**

YBOS

Definition at line 155 of file ybos.h.

**6.25.1.14   #define H\_BLOCK\_SIZE 0**

YBOS

Definition at line 154 of file ybos.h.

**6.25.1.15   #define H\_HEAD\_LEN 2**

YBOS

Definition at line 156 of file ybos.h.

**6.25.1.16   #define H\_START 3**

YBOS

Definition at line 157 of file ybos.h.

#### 6.25.1.17 `#define I1_BKTYPE 8`

Signed Integer 1 byte

Definition at line 334 of file ybos.h.

Referenced by `update_odb()`, and `ybk_iterate()`.

#### 6.25.1.18 `#define I2_BKTYPE 1`

Signed Integer 2 bytes

Definition at line 329 of file ybos.h.

Referenced by `update_odb()`.

#### 6.25.1.19 `#define I4_BKTYPE 3`

Signed Integer 4bytes

Definition at line 331 of file ybos.h.

#### 6.25.1.20 `#define MAX_BKTYPE I1_BKTYPE+1`

delimiter

Definition at line 335 of file ybos.h.

Referenced by `ybk_find()`, `ybk_list()`, and `ybk_locate()`.

#### 6.25.1.21 `#define YB_ADD_RUN 1`

YBOS

Definition at line 171 of file ybos.h.

#### 6.25.1.22 `#define YB_BANKLIST_MAX 32`

maximum number of banks to be found by the `ybk_list()(p.??)` or `bk_list()(p.??)`

Definition at line 123 of file ybos.h.

Referenced by `ybk_list()`.

**6.25.1.23 #define YB\_COMPLETE 1**

YBOS

Definition at line 167 of file ybos.h.

**6.25.1.24 #define YB\_INCOMPLETE 2**

YBOS

Definition at line 168 of file ybos.h.

**6.25.1.25 #define YB\_NO\_RECOVER -1**

YBOS

Definition at line 169 of file ybos.h.

**6.25.1.26 #define YB\_NO\_RUN 0**

YBOS

Definition at line 170 of file ybos.h.

**6.25.1.27 #define YB\_STRING\_BANKLIST\_MAX  
YB\_BANKLIST\_MAX \* 4**

to be used for xbk\_list()

Definition at line 125 of file ybos.h.

**6.25.1.28 #define YBOS\_BUFFER\_SIZE 3\*(YBOS\_-  
PHYREC\_SIZE<<2) + MAX\_EVENT\_SIZE +  
128**

in BYTES

Definition at line 121 of file ybos.h.

**6.25.1.29 #define YBOS\_HEADER\_LENGTH 4**

Definition at line 120 of file ybos.h.

6.25.1.30 `#define YBOS_PHYREC_SIZE 8192`

`I*4`

Definition at line 118 of file ybos.h.

## 6.26 YBOS Macros

### Defines

- #define **SWAP\_D2WORD**(\_d2w)
- #define **EVID\_TRINAT**
- #define **YBOS\_EVID\_BANK**(\_\_a, \_\_b, \_\_c, \_\_d, \_\_e)
- #define **MIDAS\_EVID\_BANK**(\_\_a, \_\_b, \_\_c, \_\_d, \_\_e)

### 6.26.1 Define Documentation

#### 6.26.1.1 #define EVID\_TRINAT

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

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

Omitting the declaration of the pre-processor flag the EVID\_TRINAT is taken by default see **Midas build options and operation considerations**(p.??).

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

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

```
// check if EVID is present if so display its content
if ((status = ybk_find (pybos, "EVID", &bklen, &bktyp, (void *)&pybk)) == YB_SUCCESS)
{
    pdata = (DWORD *)((YBOS_BANK_HEADER *)pybk + 1);
    pevent->event_id      = YBOS_EVID_EVENT_ID(pdata);
    pevent->trigger_mask  = YBOS_EVID_TRIGGER_MASK(pdata);
    pevent->serial_number = YBOS_EVID_SERIAL(pdata);
    pevent->time_stamp    = YBOS_EVID_TIME(pdata);
    pevent->data_size     = pybk->length;
}
```

The current type of EVID bank are:

EVID\_TRINAT Specific for Trinat experiment.

```
ybk_create((DWORD *)pevent, "EVID", I4_BKTYPE, (DWORD *)&pbkdat));
*((WORD *)pbkdat) = EVENT_ID(pevent);      ((WORD *)pbkdat)++;
```

```

*((WORD *)pbkdat) = TRIGGER_MASK(pevent); ((WORD *)pbkdat)++;
*(pbkdat)++ = SERIAL_NUMBER(pevent);
*(pbkdat)++ = TIME_STAMP(pevent);
*(pbkdat)++ = gbl_run_number;           // run number

```

EVID\_TWIST Specific to Twist Experiment (Triumph).

```

ybk_create((DWORD *)pevent, "EVID", I4_BKTYPE, &pbkdat);
*((WORD *)pbkdat) = EVENT_ID(pevent); ((WORD *)pbkdat)++;
*((WORD *)pbkdat) = TRIGGER_MASK(pevent); ((WORD *)pbkdat)++;
*(pbkdat)++ = SERIAL_NUMBER(pevent);
*(pbkdat)++ = TIME_STAMP(pevent);
*(pbkdat)++ = gbl_run_number;           // run number
*(pbkdat)++ = *((DWORD *)frontend_name); // frontend name
ybk_close((DWORD *)pevent, pbkdat);

```

Definition at line 263 of file ybos.h.

#### 6.26.1.2 #define MIDAS\_EVID\_BANK(\_\_a, \_\_b, \_\_c, \_\_d, \_\_e)

**Value:**

```

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

```

pevt Evt# id/msk serial run#

Definition at line 307 of file ybos.h.

#### 6.26.1.3 #define SWAP\_D2WORD(\_d2w)

**Value:**

```

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

```

word swap (I4=I21I22 -> I4=I22I21

Definition at line 196 of file ybos.h.

**6.26.1.4** `#define YBOS_EVID_BANK(__a, __b, __c, __d, __e)`

**Value:**

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

pevt Evt# id/msk serial run#

Definition at line 293 of file ybos.h.

**6.26.1.5** `#define YBOS_EVID_EVENT_ID(e) *((WORD *) (e)+1)`

Definition at line 267 of file ybos.h.

**6.26.1.6** `#define YBOS_EVID_EVENT_NB(e) *((DWORD *) (e)+1)`

Definition at line 272 of file ybos.h.

**6.26.1.7** `#define YBOS_EVID_RUN_NUMBER(e) *((DWORD *) (e)+3)`

Definition at line 271 of file ybos.h.

**6.26.1.8** `#define YBOS_EVID_SERIAL(e) *((DWORD *) (e)+1)`

Definition at line 269 of file ybos.h.

**6.26.1.9** `#define YBOS_EVID_TIME(e) *((DWORD *) (e)+2)`

Definition at line 270 of file ybos.h.



```
6.26.1.10 #define YBOS_EVID_TRIGGER_MASK(e)  
          *((WORD *)(e)+0)
```

Definition at line 268 of file ybos.h.

## 6.27 YBOS error code

### Defines

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

### 6.27.1 Define Documentation

#### 6.27.1.1 `#define YB_BANK_NOT_FOUND -101`

Bank not found

Definition at line 139 of file ybos.h.

Referenced by `ybk_find()`, and `ybk_locate()`.

#### 6.27.1.2 `#define YB_DONE 2`

Operation complete

Definition at line 137 of file ybos.h.

#### 6.27.1.3 `#define YB_EVENT_NOT_SWAPPED 2`

Not swapped

Definition at line 136 of file ybos.h.

#### 6.27.1.4 `#define YB_NOMORE_SLOT -103`

No more space for fragment

Definition at line 141 of file ybos.h.

#### 6.27.1.5 `#define YB_SUCCESS 1`

Ok

Definition at line 135 of file ybos.h.

Referenced by `ybk_find()`.

#### **6.27.1.6    `#define YB_SWAP_ERROR -102`**

Error swapping

Definition at line 140 of file ybos.h.

#### **6.27.1.7    `#define YB_UNKNOWN_FORMAT -104`**

Unknown format (see **YBOS format**(p.??))

Definition at line 142 of file ybos.h.

#### **6.27.1.8    `#define YB_WRONG_BANK_TYPE -100`**

Wrong bank type (see **YBOS Bank Types**(p.??))

Definition at line 138 of file ybos.h.

Referenced by `ybk_find()`, `ybk_iterate()`, `ybk_list()`, and `ybk_locate()`.

## 6.28 YBOS Bank Functions (**ybk\_**xxx)

### Functions

- void **ybk\_init** (DWORD \*plrl)
- void **ybk\_create** (DWORD \*plrl, char \*bkname, DWORD bktype, void \*pbkdat)
- INT **ybk\_close** (DWORD \*plrl, void \*pbkdat)
- INT **ybk\_size** (DWORD \*plrl)
- INT **ybk\_list** (DWORD \*plrl, char \*bklist)
- INT **ybk\_find** (DWORD \*plrl, char \*bkname, DWORD \*bklen, DWORD \*bktype, void \*\*pbk)
- INT **ybk\_locate** (DWORD \*plrl, char \*bkname, void \*pdata)
- INT **ybk\_iterate** (DWORD \*plrl, YBOS\_BANK\_HEADER \*\*pybkh, void \*\*pdata)

### 6.28.1 Function Documentation

#### 6.28.1.1 INT **ybk\_close** (DWORD \* *plrl*, void \* *pbkdat*)

Close the YBOS bank previously created by **ybk\_create**(p.??).

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

#### Parameters:

*plrl* pointer to current composed event.

*pbkdat* pointer to the current data.

#### Returns:

number number of bytes contained in bank.

Definition at line 539 of file ybos.c.

#### 6.28.1.2 void **ybk\_create** (DWORD \* *plrl*, char \* *bkname*, DWORD *bktype*, void \* *pbkdat*)

Define the following memory area to be a YBOS bank with the given attribute. See **YBOS bank examples**(p.??).

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

**Parameters:**

**plrl** pointer to the first DWORD of the event area.  
**bkname** name to be assigned to the created bank (max 4 char)  
**bktype** YBOS Bank Types(p.??) of the values for the entire created bank.  
**pbkdat** return pointer to the first empty data location.

**Returns:**

void

Definition at line 426 of file ybos.c.

### 6.28.1.3 INT ybk\_find (DWORD \* **plrl**, char \* **bkname**, DWORD \* **bklen**, DWORD \* **bktype**, void \*\* **pbk**)

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

**Parameters:**

**plrl** pointer to the area of event.  
**bkname** name of the bank to be located.  
**bklen** returned length in 4bytes unit of the bank.  
**bktype** returned bank type.  
**pbk** pointer to the first data of the found bank.

**Returns:**

YB\_SUCCESS, YB\_BANK\_NOT\_FOUND, YB\_WRONG\_BANK\_TYPE

Definition at line 636 of file ybos.c.

### 6.28.1.4 void ybk\_init (DWORD \* **plrl**)

Initializes an event for YBOS banks structure.

Before banks can be created in an event, **ybk\_init()**(p.??) has to be called first. See **YBOS bank examples**(p.??).

**Parameters:**

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

**Returns:**

void

Definition at line 396 of file ybos.c.

**6.28.1.5** INT ybk\_iterate (DWORD \* *plrl*,  
YBOS\_BANK\_HEADER \*\* *pybkh*, void \*\* *pdata*)

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

**Parameters:**

*plrl* pointer to the area of event.

*pybkh* pointer to the YBOS bank header.

*pdata* pointer to the first data of the current bank.

**Returns:**

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

Definition at line 723 of file ybos.c.

Referenced by update\_odb().

**6.28.1.6** INT ybk\_list (DWORD \* *plrl*, char \* *bklist*)

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

The **bk\_list()**(p.??) has to be a predefined string of max size of YB\_STRING\_BANKLIST\_MAX.

**Parameters:**

*plrl* pointer to the area of event

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

**Returns:**

number of banks found in this event.

Definition at line 586 of file ybos.c.

**6.28.1.7 INT ybk\_locate (DWORD \* *plrl*, char \* *bkname*, void \* *pdata*)**

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

**Parameters:**

*plrl* pointer to the area of event

*bkname* name of the bank to be located.

*pdata* pointer to the first data of the located bank.

**Returns:**

Number of DWORD in bank or YB\_BANK\_NOT\_FOUND, YB\_WRONG\_BANK\_TYPE (<0)

Definition at line 684 of file ybos.c.

**6.28.1.8 INT ybk\_size (DWORD \* *plrl*)**

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

**Parameters:**

*plrl* pointer to the area of event

**Returns:**

number of bytes contained in data area of the event

Definition at line 571 of file ybos.c.

## 6.29 Midas Common Functions (cm\_XXX)

### 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** (HANDLE hDB, INT pid)
- INT **cm\_check\_client** (HANDLE hDB, HANDLE hKeyClient)
- INT **cm\_set\_client\_info** (HANDLE hDB, HANDLE \*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, HANDLE \*hConn)
- INT **cm\_disconnect\_client** (HANDLE hConn, BOOL bShutdown)
- INT **cm\_disconnect\_experiment** (void)
- INT **cm\_set\_experiment\_database** (HANDLE hDB, HANDLE hKeyClient)
- INT **cm\_get\_experiment\_database** (HANDLE \*hDB, HANDLE \*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** (HANDLE hDB, char \*client\_name, DWORD \*timeout, DWORD \*last)
- INT **cm\_register\_transition** (INT transition, INT(\*func)(INT, char \*))
- 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\_flag*, INT *debug\_flag*)
- INT **cm\_yield** (INT *millisec*)
- INT **cm\_execute** (char \**command*, char \**result*, INT *bufsize*)
- INT **cm\_shutdown** (char \**name*, BOOL *bUnique*)
- INT **cm\_exist** (char \**name*, BOOL *bUnique*)
- INT **cm\_cleanup** (char \**client\_name*, BOOL *ignore\_timeout*)

## 6.29.1 Function Documentation

### 6.29.1.1 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

Definition at line 1613 of file *midas.c*.

Referenced by *al\_trigger\_alarm()*, and *cm\_transition()*.

### 6.29.1.2 INT **cm\_check\_client** (HANDLE *hDB*, HANDLE *hKeyClient*)

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

**Parameters:**

*hDB* Handle to online database

*hKeyClient* Handle to client key

**Returns:**

CM\_SUCCESS, CM\_NO\_CLIENT

Definition at line 1881 of file *midas.c*.

Referenced by *cm\_set\_client\_info()*.

### 6.29.1.3 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 calls the registered handler if the transition should be done and then actually does the transition.

#### Returns:

CM\_SUCCESS, <error> Error from `cm_transition()` (p.??)

Definition at line 3355 of file `midas.c`.

Referenced by `scheduler()`.

### 6.29.1.4 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 removal is done via `cm_watchdog()`.

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

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

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

#### Parameters:

*client\_name* Client name, if zero check all clients

*ignore\_timeout* If TRUE, ignore a possible increased timeout defined by each client.

**Returns:**

CM\_SUCCESS

Definition at line 4989 of file midas.c.

Referenced by main().

#### 6.29.1.5 INT cm\_connect\_client (char \* *client\_name*, HANDLE \* *hConn*)

Connect to a MIDAS client of the current experiment

**For Parameter only.**

*client\_name* Name of client to connect to. This name is set by the other client via the cm\_connect\_experiment call.

*hConn* Connection handle

**Returns:**

CM\_SUCCESS, CM\_NO\_CLIENT

Definition at line 2664 of file midas.c.

#### 6.29.1.6 INT cm\_connect\_experiment (char \* *host\_name*, char \* *exp\_name*, char \* *client\_name*, void(\* *func*)(char \*))

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

**Attention:**

All MIDAS applications should evaluate the MIDAS\_SERVER\_HOST and MIDAS\_EXPT\_NAME environment variables as defaults to the host name and experiment name (see **Environment variables**(p.??)). For that purpose, the function **cm\_get\_environment**(p.??) should be called prior to **cm\_connect\_experiment**(p.??). If command line parameters -h and -e are used, the evaluation should be done between **cm\_get\_environment**(p.??) and **cm\_connect\_experiment**(p.??). The function **cm\_disconnect\_experiment**(p.??) must be called before a MIDAS application exits.

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

```

char host_name[256],exp_name[32];

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

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

status = cm_connect_experiment(host_name, exp_name, "Test", NULL);
if (status != CM_SUCCESS)
    return 1;
...do operations...
cm_disconnect_experiment();
}

```

**Parameters:**

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

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

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

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

**Returns:**

CM\_SUCCESS, CM\_UNDEF\_EXP, CM\_SET\_ERROR, RPC\_NET\_ERROR

CM\_VERSION\_MISMATCH MIDAS library version different on local and remote computer

Definition at line 2297 of file midas.c.

Referenced by main().

**6.29.1.7** INT **cm\_connect\_experiment1** (char \* *host\_name*, char \* *exp\_name*, char \* *client\_name*, void(\* *func*)(char \*), INT *odb\_size*, DWORD *watchdog\_timeout*)

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

**For internal use only.**

Definition at line 2321 of file midas.c.

Referenced by cm\_connect\_experiment(), and main().

**6.29.1.8** INT **cm\_delete\_client\_info** (HANDLE *hDB*, INT *pid*)

Delete client info from database

**Parameters:**

*hDB* Database handle

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

**Returns:**

CM\_SUCCESS

Definition at line 1830 of file midas.c.

Referenced by cm\_check\_client(), cm\_cleanup(), and cm\_disconnect\_experiment().

**6.29.1.9** INT **cm\_disconnect\_client** (HANDLE *hConn*, BOOL *bShutdown*)

Disconnect from a MIDAS client

**Parameters:**

*hConn* Connection handle obtained via cm\_connect\_client()(p.??)

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

**Returns:**

see `rpc_client_disconnect()`

Definition at line 2723 of file `midas.c`.

**6.29.1.10 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()`(p. ??) for example.

**Returns:**

CM\_SUCCESS

Definition at line 2738 of file `midas.c`.

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

**6.29.1.11 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

Definition at line 4061 of file `midas.c`.

**6.29.1.12 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

Definition at line 4915 of file midas.c.

Referenced by main().

### 6.29.1.13 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

Definition at line 2127 of file midas.c.

Referenced by bm\_open\_buffer().

### 6.29.1.14 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**(p.??)). The usual way is that the host name and experiment name are first derived from the environment variables MIDAS\_SERVER\_HOST and MIDAS\_EXPT\_NAME. They can then be superseded by command line parameters with -h and -e flags.

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

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

    // parse command line parameters
```

```

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

```

**Parameters:**

*host\_name* Contents of MIDAS\_SERVER\_HOST environment variable.

*host\_name\_size* string length

*exp\_name* Contents of MIDAS\_EXPT\_NAME environment variable.

*exp\_name\_size* string length

**Returns:**

CM\_SUCCESS

Definition at line 2201 of file midas.c.

Referenced by main().

#### 6.29.1.15 INT cm\_get\_experiment\_database (HANDLE \* *hDB*, HANDLE \* *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.



```

HANDLE hDB, hkeyclient;
char  name[32];
int   size;
db_get_experiment_database(&hdb, &hkeyclient);
size = sizeof(name);
db_get_value(hdb, hkeyclient, "Name", name, &size, TID_STRING, TRUE);
printf("My name is %s\n", name);

```

**Parameters:**

*hDB* Database handle.

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

**Returns:**

CM\_SUCCESS

Definition at line 2872 of file midas.c.

Referenced by al\_trigger\_alarm(), 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\_watchdog\_params(), cm\_shutdown(), cm\_transition(), el\_submit(), and main().

**6.29.1.16 INT cm\_get\_path (char \* path)**

Return the path name previously set with cm\_set\_path.

**Parameters:**

*path* Pathname

**Returns:**

CM\_SUCCESS

Definition at line 1704 of file midas.c.

Referenced by cm\_connect\_experiment1(), cm\_msg\_log(), cm\_msg\_log1(), and cm\_msg\_retrieve().

**6.29.1.17 char\* cm\_get\_version ()**

Return version number of current MIDAS library as a string

**Returns:**

version number \* 100

Definition at line 1671 of file midas.c.

Referenced by cm\_transition().

**6.29.1.18 INT cm\_get\_watchdog\_info (HANDLE *hDB*, char \* *client\_name*, DWORD \* *timeout*, DWORD \* *last*)**

Return watchdog information about specific client

**Parameters:**

*hDB* ODB handle

*client\_name* ODB client name

*timeout* Timeout for this application in seconds

*last* Last time watchdog was called in msec

**Returns:**

CM\_SUCCESS, CM\_NO\_CLIENT, DB\_INVALID\_HANDLE

Definition at line 3086 of file midas.c.

**6.29.1.19 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

Definition at line 3067 of file midas.c.

Referenced by bm\_open\_buffer(), cm\_connect\_experiment1(), cm\_set\_client\_info(), and db\_open\_database().

**6.29.1.20 INT cm\_list\_experiments (char \* *host\_name*, char *exp\_name*[MAX\_EXPERIMENT][NAME\_LENGTH])**

Connect to a MIDAS server and return all defined experiments in \*exp\_name[MAX\_EXPERIMENTS]

**Parameters:**

*host\_name* Internet host name.

*exp\_name* list of experiment names

**Returns:**

CM\_SUCCESS, RPC\_NET\_ERROR

Definition at line 2522 of file midas.c.

Referenced by cm\_select\_experiment().

**6.29.1.21 INT cm\_register\_deferred\_transition (INT *transition*,  
BOOL(\**func*)(INT, BOOL))**

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

**Parameters:**

*transition* One of TR\_xxx

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

**Returns:**

CM\_SUCCESS, <error> Error from ODB access

Definition at line 3292 of file midas.c.

**6.29.1.22 INT cm\_register\_transition (INT *transition*, INT(\**func*)(INT, char \*))**

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

Clients can register for transitions (Start/Stop/Pause/Resume) or for notifications before or after a transition occurs (Pre-start/Post-start/Pre-stop/Post-stop). The logger for example opens the logging files on pre-start and closes them on post-stop.

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()**(p. ??) 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);
    do
    {
        status = cm_yield(1000);
    } while (status != RPC_SHUTDOWN &&
            status != SS_ABORT);
    ...
}

```

**Parameters:**

*transition* Transition to register for (see )

*func* Callback function.

**Returns:**

CM\_SUCCESS

Definition at line 3236 of file midas.c.

Referenced by main().

**6.29.1.23 INT cm\_scan\_experiments (void)**

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

**Returns:**

CM\_SUCCESS

CM\_UNDEF\_EXP exptab not found and MIDAS\_DIR not set

Definition at line 1735 of file midas.c.

Referenced by cm\_connect\_experiment1(), and cm\_list\_experiments().

**6.29.1.24 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 Internet use only.**

*host\_name* Internet host name.

*exp\_name* list of experiment names

**Returns:**

CM\_SUCCESS, RPC\_NET\_ERROR

Definition at line 2624 of file midas.c.

Referenced by cm\_connect\_experiment1().

**6.29.1.25 INT cm\_set\_client\_info (HANDLE *hDB*, HANDLE \* *hKeyClient*, char \* *host\_name*, char \* *client\_name*, INT *hw\_type*, char \* *password*, DWORD *watchdog\_timeout*)**

Set client information in online database and return handle

**Parameters:**

*hDB* Handle to online database

*hKeyClient* returned key

*host\_name* server name

*client\_name* Name of this program as it will be seen by other clients.

*hw\_type* Type of byte order

*password* MIDAS password

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

**Returns:**

CM\_SUCCESS

Definition at line 1940 of file midas.c.

Referenced by cm\_connect\_experiment1().

**6.29.1.26 INT cm\_set\_experiment\_database (HANDLE *hDB*, HANDLE *hKeyClient*)**

Set the handle to the ODB for the currently connected experiment

**Parameters:**

*hDB* Database handle

*hKeyClient* Key handle of client structure

**Returns:**

CM\_SUCCESS

Definition at line 2813 of file midas.c.

Referenced by cm\_connect\_experiment1(), and cm\_disconnect\_experiment().

**6.29.1.27 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

Definition at line 1686 of file midas.c.

Referenced by cm\_connect\_experiment1().

**6.29.1.28 INT cm\_set\_watchdog\_params (BOOL *call\_watchdog*,  
DWORD *timeout*)**

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

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

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

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

**Parameters:**

*call\_watchdog* Call the `cm_watchdog` routine periodically

*timeout* Timeout for this application in ms

**Returns:**

`CM_SUCCESS`

Definition at line 2958 of file `midas.c`.

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

#### 6.29.1.29 INT `cm_shutdown` (char \* *name*, BOOL *bUnique*)

Shutdown (exit) other MIDAS client

**Parameters:**

*name* Client name or "all" for all clients

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

**Returns:**

`CM_SUCCESS`, `CM_NO_CLIENT`, `DB_NO_KEY`

Definition at line 4820 of file `midas.c`.

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

#### 6.29.1.30 INT `cm_synchronize` (DWORD \* *seconds*)

Get time from MIDAS server and set local time.

**Parameters:**

*seconds* Time in seconds

**Returns:**

`CM_SUCCESS`

Definition at line 1585 of file midas.c.

Referenced by main().

#### 6.29.1.31 INT cm\_time (DWORD \* *time*)

Get time from ss\_time on server.

##### Parameters:

*time* string

##### Returns:

CM\_SUCCESS

Definition at line 1632 of file midas.c.

Referenced by cm\_transition().

#### 6.29.1.32 INT cm\_transition (INT *transition*, INT *run\_number*, char \* *perror*, INT *strsize*, INT *async\_flag*, INT *debug\_flag*)

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

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

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

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



**Parameters:**

***transition*** TR\_START, TR\_PAUSE, TR\_RESUME or TR\_STOP.  
***run\_number*** New run number. If zero, use current run number plus one.  
***perror*** returned error string.  
***strsize*** Size of error string.  
***async\_flag*** SYNC: synchronization flag (SYNC:wait completion, ASYNC: return immediately)  
***debug\_flag*** If true output debuginfo information.

**Returns:**

CM\_SUCCESS, <error> error code from remote client

Definition at line 3454 of file midas.c.

Referenced by cm\_check\_deferred\_transition(), main(), and scheduler().

**6.29.1.33 INT cm\_yield (INT *millisec*)**

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

**Parameters:**

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

**Returns:**

CM\_SUCCESS, RPC\_SHUTDOWN

Definition at line 4007 of file midas.c.

Referenced by main(), and scheduler().

## 6.30 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** (HANDLE buffer\_handle, short int event\_id, short int trigger\_mask, INT sampling\_type, HANDLE \*request\_id, void(\*func)(HANDLE, HANDLE, **EVENT\_HEADER** \*, void \*))
- **INT bm\_remove\_event\_request** (INT buffer\_handle, INT request\_id)
- **INT bm\_delete\_request** (INT request\_id)
- **INT bm\_send\_event** (INT buffer\_handle, void \*source, INT buf\_size, INT async\_flag)
- **INT bm\_flush\_cache** (INT buffer\_handle, INT async\_flag)
- **INT bm\_receive\_event** (INT buffer\_handle, void \*destination, INT \*buf\_size, INT async\_flag)
- **INT bm\_skip\_event** (INT buffer\_handle)
- **INT bm\_push\_event** (char \*buffer\_name)
- **INT bm\_check\_buffers** ()
- **INT bm\_empty\_buffers** ()

### 6.30.1 Function Documentation

#### 6.30.1.1 INT bm\_check\_buffers ()

Check if any requested event is waiting in a buffer

#### Returns:

TRUE More events are waiting  
FALSE No more events are waiting

Definition at line 7299 of file midas.c.

Referenced by cm\_yield().

**6.30.1.2 INT bm\_close\_all\_buffers (void)**

Close all open buffers

**Returns:**

BM\_SUCCESS

Definition at line 4555 of file midas.c.

Referenced by cm\_disconnect\_experiment(), and cm\_set\_client\_info().

**6.30.1.3 INT bm\_close\_buffer (INT *buffer\_handle*)**

Closes an event buffer previously opened with **bm\_open\_buffer()**(p. ??).

**Parameters:**

*buffer\_handle* buffer handle

**Returns:**

BM\_SUCCESS, BM\_INVALID\_HANDLE

Definition at line 4446 of file midas.c.

Referenced by bm\_close\_all\_buffers(), and source\_unbooking().

**6.30.1.4 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 of the data part of the event in bytes  
*serial* serial number

**Returns:**

BM\_SUCCESS

Definition at line 5521 of file midas.c.

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

**6.30.1.5 INT `bm_delete_request` (INT *request\_id*)**

Deletes an event request previously done with `bm_request_event()`(p.??). 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()`(p.??), all event requests from that buffer are deleted automatically

**Parameters:**

*request\_id* request identifier given by `bm_request_event()`(p.??)

**Returns:**

BM\_SUCCESS, BM\_INVALID\_HANDLE

Definition at line 5828 of file midas.c.

Referenced by `bm_close_buffer()`, and `source_unbooking()`.

**6.30.1.6 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()`(p.??)). When changes to the hardware are made in the experience, the consumer will then still analyze old events before any new event which reflects the hardware change. Users can be fooled by looking at histograms which reflect the hardware change many seconds after they have been made.

To overcome this potential problem, the analyzer can call `bm_empty_buffers()`(p.??) 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(p.??)` reduces this effect by flushing all buffers once every second.

**Returns:**

BM\_SUCCESS

Definition at line 7613 of file `midas.c`.

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

### 6.30.1.7 INT `bm_flush_cache` (INT *buffer\_handle*, INT *async\_flag*)

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

**Parameters:**

*buffer\_handle* Buffer handle obtained via `bm_open_buffer(p.??)`  
*async\_flag* Synchronous/asynchronous flag. If FALSE, the function blocks if the buffer has not enough free space to receive the full cache. If TRUE, the function returns immediately with a value of BM\_ASYNC\_RETURN without writing the cache.

**Returns:**

BM\_SUCCESS, BM\_INVALID\_HANDLE  
 BM\_ASYNC\_RETURN Routine called with `async_flag == TRUE` and buffer has not enough space to receive cache  
 BM\_NO\_MEMORY Event is too large for network buffer or event buffer. One has to increase MAX\_EVENT\_SIZE or EVENT\_BUFFER\_SIZE in `midas.h(p.??)` and recompile.

Definition at line 6254 of file `midas.c`.

Referenced by `bm_send_event()`, `main()`, `scheduler()`, and `send_event()`.

### 6.30.1.8 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 4158 of file midas.c.

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

### 6.30.1.9 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`(p.??) 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 conflicts with an existing buffer of different size  
 BM\_INVALID\_PARAM Invalid parameter

Definition at line 4223 of file midas.c.

Referenced by cm\_msg(), cm\_msg1(), cm\_msg\_register(), main(), and source\_booking().

#### 6.30.1.10 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

Definition at line 7019 of file midas.c.

Referenced by bm\_check\_buffers().

#### 6.30.1.11 INT bm\_receive\_event (INT *buffer\_handle*, void \* *destination*, INT \* *buf\_size*, INT *async\_flag*)

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

It can be used in analysis systems which actively receive events, rather than using callbacks. A analysis package could for example contain its own command line interface. A command like "receive 1000 events" could make it necessary to call **bm\_receive\_event**(p.??) 1000 times in a row to receive these events and then return back to the command line prompt. The according **bm\_request\_event**(p.??) call contains NULL as the callback routine to indicate that **bm\_receive\_event**(p.??) 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;
    HANDLE 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\_flag* Synchronous/asynchronous flag. If FALSE, the function blocks if no event is available. If TRUE, the function returns immediately with a value of BM\_ASYNC\_RETURN without receiving any event.

**Returns:**

BM\_SUCCESS, BM\_INVALID\_HANDLE

BM\_TRUNCATED The event is larger than the destination buffer and was therefore truncated

BM\_ASYNC\_RETURN No event available

Definition at line 6627 of file midas.c.

Referenced by handFlush(), and source\_scan().



### 6.30.1.12 INT **bm\_remove\_event\_request** (INT *buffer\_handle*, INT *request\_id*)

Delete a previously placed request for a specific event type in the client structure of the buffer referenced by *buffer\_handle*.

#### Parameters:

*buffer\_handle* Handle to the buffer where the request should be placed in  
*request\_id* Request id returned by **bm\_request\_event**

#### Returns:

BM\_SUCCESS, BM\_INVALID\_HANDLE, BM\_NOT\_FOUND, RPC\_NET\_ERROR

Definition at line 5757 of file midas.c.

Referenced by **bm\_delete\_request()**.

### 6.30.1.13 INT **bm\_request\_event** (HANDLE *buffer\_handle*, short int *event\_id*, short int *trigger\_mask*, INT *sampling\_type*, HANDLE \* *request\_id*, void(\**func*)(HANDLE, HANDLE, EVENT\_HEADER \*, void \*))

Place an event request based on certain characteristics. Multiple event requests can be placed for each buffer, which are later identified by their request ID. They can contain different callback routines. Example see **bm\_open\_buffer()**(p.??) and **bm\_receive\_event()**(p.??)

#### Parameters:

*buffer\_handle* buffer handle obtained via **bm\_open\_buffer()**(p.??)  
*event\_id* event ID for requested events. Use EVENTID\_ALL to receive events with any ID.  
*trigger\_mask* trigger mask for requested events. The requested events must have at least one bit in its trigger mask common with the requested trigger mask. Use TRIGGER\_ALL to receive events with any trigger mask.  
*sampling\_type* specifies how many events to receive. A value of GET\_ALL receives all events which match the specified event ID and trigger mask. If the events are consumed slower than produced, the producer is automatically slowed down. A value of GET\_SOME receives as much events as possible without slowing down the producer. GET\_ALL is typically used by the logger, while GET\_SOME is typically used by analyzers.

**request\_id** request ID returned by the function. This ID is passed to the callback routine and must be used in the **bm\_delete\_request()**(p.??) routine.

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

**Returns:**

BM\_SUCCESS, BM\_INVALID\_HANDLE

BM\_NO\_MEMORY too many requests. The value MAX\_EVENT\_REQUESTS in **midas.h**(p.??) should be increased.

Definition at line 5685 of file midas.c.

Referenced by **cm\_msg\_register()**, and **source\_booking()**.

#### 6.30.1.14 INT **bm\_send\_event** (INT *buffer\_handle*, void \* *source*, INT *buf\_size*, INT *async\_flag*)

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

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

// set first byte of event
*(event+sizeof(EVENT_HEADER)) = <...>
#include <stdio.h>
#include "midas.h"
main()
{
    INT status, i;
    HNDLE hbuf;
    char event[1000];
    status = cm_connect_experiment("", "Sample", "Producer", NULL);
    if (status != CM_SUCCESS)
        return 1;
    bm_open_buffer(EVENT_BUFFER_NAME, 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()**(p.??)  
*source* Address of event buffer  
*buf\_size* Size of event including event header in bytes  
*async\_flag* Synchronous/asynchronous flag. If FALSE, the function blocks if the buffer has not enough free space to receive the event. If TRUE, the function returns immediately with a value of BM\_ASYNC\_RETURN without writing the event to the buffer

**Returns:**

BM\_SUCCESS, BM\_INVALID\_HANDLE, BM\_INVALID\_PARAM  
 BM\_ASYNC\_RETURN Routine called with *async\_flag* == TRUE and buffer has not enough space to receive event  
 BM\_NO\_MEMORY Event is too large for network buffer or event buffer. One has to increase MAX\_EVENT\_SIZE or EVENT\_BUFFER\_SIZE in **midas.h**(p.??) and recompile.

Definition at line 5892 of file **midas.c**.

Referenced by **cm\_msg()**, **cm\_msg1()**, **rpc\_send\_event()**, and **send\_event()**.

#### 6.30.1.15 INT **bm\_set\_cache\_size** (INT *buffer\_handle*, INT *read\_size*, INT *write\_size*)

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

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

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

**Parameters:**

*buffer\_handle* buffer handle obtained via **bm\_open\_buffer()**(p.??)  
*read\_size* cache size for reading events in bytes, zero for no cache  
*write\_size* cache size for writing events in bytes, zero for no cache

**Returns:**

BM\_SUCCESS, BM\_INVALID\_HANDLE, BM\_NO\_MEMORY, BM\_INVALID\_PARAM

Definition at line 5429 of file midas.c.

Referenced by main().

**6.30.1.16 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

Definition at line 6972 of file midas.c.

## 6.31 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 \**filename*, INT *line*, const char \**routine*, const char \**format*,...)
- INT **cm\_msg1** (INT *message\_type*, char \**filename*, INT *line*, const char \**facility*, const char \**routine*, const char \**format*,...)
- INT **cm\_msg\_register** (void(\**func*)(HANDLE, HANDLE, EVENT\_HEADER \*, void \*))
- INT **cm\_msg\_retrieve** (INT *n\_message*, char \**message*, INT \**buf\_size*)

### 6.31.1 Function Documentation

#### 6.31.1.1 INT cm\_get\_error (INT *code*, char \* *string*)

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

##### Parameters:

*code* Error code as defined in `midas.h`(p.??)

*string* Error string

##### Returns:

CM\_SUCCESS

Definition at line 940 of file `midas.c`.

Referenced by `cm_connect_experiment()`.

#### 6.31.1.2 INT cm\_msg (INT *message\_type*, char \* *filename*, INT *line*, const char \* *routine*, const char \* *format*, ...)

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

**Attention:**

Do not add the "\n" escape carriage control at the end of the formatted 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(p.??)).

*filename* Name of source file where error occurred

*line* Line number where error occurred

*routine* Routine name.

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

**Returns:**

CM\_SUCCESS

Definition at line 1192 of file midas.c.

Referenced by al\_trigger\_alarm(), bk\_list(), bm\_close\_buffer(), bm\_flush\_cache(), bm\_open\_buffer(), bm\_push\_event(), bm\_receive\_event(), bm\_remove\_event\_request(), bm\_request\_event(), bm\_send\_event(), bm\_set\_cache\_size(), bm\_skip\_event(), 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\_set\_client\_info(), 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\_find\_key(), db\_get\_data(), db\_get\_data\_index(), db\_get\_key(), db\_get\_key\_info(), db\_get\_key\_time(), db\_get\_record(), db\_get\_value(), db\_load(), db\_lock\_database(), db\_open\_database(), db\_open\_record(), db\_paste(), db\_protect\_database(), db\_save(), db\_save\_struct(), db\_set\_data(), db\_set\_data\_index(), db\_set\_record(), db\_set\_value(), db\_unlock\_database(), dm\_buffer\_create(), el\_submit(), handFlush(), interrupt\_routine(), main(), rpc\_flush\_event(), rpc\_register\_functions(), rpc\_send\_event(), rpc\_set\_option(), scheduler(), send\_event(), source\_booking(), source\_scan(), source\_unbooking(), tr\_prestart(), update\_odb(), and ybk\_list().

### 6.31.1.3 INT `cm_msg1` (INT *message\_type*, char \* *filename*, INT *line*, const char \* *facility*, const char \* *routine*, const char \* *format*, ...)

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

**For Attention only.**

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

```
...
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(p.??)`.  
*filename* Name of source file where error occurred  
*line* Line number where error occurred  
*facility* Logging file name  
*routine* Routine name  
*format* message to printout, ... Parameters like for `printf()`

**Returns:**

`CM_SUCCESS`

Definition at line 1310 of file `midas.c`.

### 6.31.1.4 INT `cm_msg_log` (INT *message\_type*, const char \* *message*)

Write message to logging file. Called by `cm_msg`.

**Attention:**

May burn your fingers

**Parameters:**

*message\_type* Message type  
*message* Message string

**Returns:**

`CM_SUCCESS`

Definition at line 1003 of file midas.c.

Referenced by `cm_msg()`.

#### 6.31.1.5 INT `cm_msg_log1` (INT *message\_type*, const char \**message*, const char \**facility*)

Write message to logging file. Called by `cm_msg()`(p.??).

For ~~Intermet~~ **Intermet** case only.

*message\_type* Message type

*message* Message string

*facility* Message facility, filename in which messages will be written

**Returns:**

CM\_SUCCESS

Definition at line 1071 of file midas.c.

Referenced by `cm_msg1()`.

#### 6.31.1.6 INT `cm_msg_register` (void(\* *func*)(HANDLE, HANDLE, EVENT\_HEADER \*, void \*))

Register a dispatch function for receiving system messages.

- example code from `mlxspeaker.c`

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

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

int main(int argc, char *argv[])
{
    ...
}
```



```

    // now connect to server
    status = cm_connect_experiment(host_name, exp_name, "Speaker", NULL);
    if (status != CM_SUCCESS)
        return 1;
    // Register callback for messages
    cm_msg_register(receive_message);
    ...
}

```

**Parameters:**

*func* Dispatch function.

**Returns:**

CM\_SUCCESS or bm\_open\_buffer and bm\_request\_event return status

Definition at line 1446 of file midas.c.

### 6.31.1.7 INT cm\_msg\_retrieve (INT *n\_message*, char \* *message*, INT \* *buf\_size*)

Retrieve old messages from log file

**Parameters:**

*n\_message* Number of messages to retrieve

*message* *buf\_size* bytes of messages, separated by characters. The returned number of bytes is normally smaller than the initial *buf\_size*, since only full lines are returned.

\**buf\_size* Size of message buffer to fill

**Returns:**

CM\_SUCCESS

Definition at line 1478 of file midas.c.

### 6.31.1.8 INT cm\_set\_msg\_print (INT *system\_mask*, INT *user\_mask*, int(\* *func*)(const char \*))

Set message masks. When a message is generated by calling **cm\_msg()**(p. ??), it can go to two destinations. First a user defined callback routine and second to the "SYSMSG" buffer.

A user defined callback receives all messages which satisfy the *user\_mask*.

```

int message_print(const char *msg)
{

```

```
char str[160];

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

**Parameters:**

***system\_mask*** Bit masks for MERROR, MINFO etc. to send system messages.

***user\_mask*** Bit masks for MERROR, MINFO etc. to send messages to the user callback.

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

**Returns:**

CM\_SUCCESS

Definition at line 985 of file midas.c.

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

## 6.32 Midas Bank Functions (bk\_XXX)

### Functions

- void **bk\_init** (void \*event)
- void **bk\_init32** (void \*event)
- INT **bk\_size** (void \*event)
- void **bk\_create** (void \*event, const char \*name, WORD type, void \*pdata)
- INT **bk\_close** (void \*event, void \*pdata)
- INT **bk\_list** (void \*event, char \*bklist)
- INT **bk\_locate** (void \*event, const char \*name, void \*pdata)
- INT **bk\_find** (**BANK\_HEADER** \*pbkh, const char \*name, DWORD \*bklen, DWORD \*bktype, void \*\*pdata)
- INT **bk\_iterate** (void \*event, **BANK** \*\*pbk, void \*pdata)
- INT **bk\_swap** (void \*event, BOOL force)

### 6.32.1 Function Documentation

#### 6.32.1.1 INT **bk\_close** (void \* *event*, void \* *pdata*)

Close the Midas bank previously created by **bk\_create**(p.??). The data pointer *pdata* must be obtained by **bk\_create**(p.??) and used as an address to fill a bank. It is incremented with every value written to the bank and finally points to a location just after the last byte of the bank. It is then passed to **bk\_close**(p.??) to finish the bank creation

#### Parameters:

*event* pointer to current composed event  
*pdata* pointer to the data

#### Returns:

number of bytes contained in bank

Definition at line 12511 of file midas.c.

#### 6.32.1.2 void **bk\_create** (void \* *event*, const char \* *name*, WORD *type*, void \* *pdata*)

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

```

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

```

**Parameters:**

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

**Returns:**

void

Definition at line 12381 of file midas.c.

**6.32.1.3** INT **bk\_find** (BANK\_HEADER \* *pbkh*, const char \* *name*, DWORD \* *bklen*, DWORD \* *bktype*, void \*\* *pdata*)

Finds a MIDAS bank of given name inside an event.

**Parameters:**

*pbkh* pointer to current composed event  
*name* bank name to look for  
*bklen* number of elements in bank  
*bktype* bank type, one of TID\_XXX  
*pdata* pointer to data area of bank, NULL if bank not found

**Returns:**

1 if bank found, 0 otherwise

Definition at line 12673 of file midas.c.

**6.32.1.4** void **bk\_init** (void \* *event*)

Initializes an event for Midas banks structure. Before banks can be created in an event, **bk\_init**(p.??) has to be called first.

**Parameters:**

*event* pointer to the area of event

Definition at line 12299 of file midas.c.

**6.32.1.5 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()**(p.??) has to be called first.

**Parameters:**

*event* pointer to the area of event

**Returns:**

void

Definition at line 12339 of file midas.c.

**6.32.1.6 INT bk\_iterate (void \* event, BANK \*\* pbk, void \* pdata)**

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

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

where type is a TID\_XXX value and data\_size the size of the bank in bytes.

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

**Parameters:**

*event* Pointer to data area of event.

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

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

**Returns:**

Size of bank in bytes

Definition at line 12762 of file midas.c.

Referenced by bk\_list(), and update\_odb().

### 6.32.1.7 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(p.??): 32 banks max).

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

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

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

**Parameters:**

*event* pointer to current composed event

*bklist* returned ASCII string, has to be booked with STRING\_BANKLIST\_MAX.

**Returns:**

number of bank found in this event.

Definition at line 12571 of file midas.c.

### 6.32.1.8 INT bk\_locate (void \* *event*, const char \* *name*, void \* *pdata*)

Locates a MIDAS bank of given name inside an event.

**Parameters:**

*event* pointer to current composed event

**name** bank name to look for

**pdata** pointer to data area of bank, NULL if bank not found

**Returns:**

number of values inside the bank

Definition at line 12619 of file midas.c.

### 6.32.1.9 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**(p.??) plus the size of the event header (sizeof(EVENT\_HEADER)).

**Parameters:**

**event** pointer to the area of event

**Returns:**

number of bytes contained in data area of event

Definition at line 12353 of file midas.c.

### 6.32.1.10 INT **bk\_swap** (void \* *event*, BOOL *force*)

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

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

**Parameters:**

**event** pointer to data area of event

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

**Returns:**

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

Definition at line 12834 of file midas.c.

Referenced by **eb\_mfragment\_add**(), and **source\_scan**().

## 6.33 Midas Alarm Functions (al\_XXX)

### Functions

- **INT al\_trigger\_alarm** (char \*alarm\_name, char \*alarm\_message, char \*default\_class, char \*cond\_str, INT type)

### 6.33.1 Function Documentation

#### 6.33.1.1 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

Definition at line 15775 of file midas.c.



## 6.34 Midas History Functions (hs\_XXX)

### Functions

- INT **hs\_set\_path** (char \*path)
- INT **hs\_open\_file** (DWORD *ltime*, char \**suffix*, INT *mode*, int \**fh*)

### 6.34.1 Function Documentation

#### 6.34.1.1 INT **hs\_open\_file** (DWORD *ltime*, char \* *suffix*, INT *mode*, int \* *fh*)

Open history file belonging to certain date. Internal use only.

##### Parameters:

*ltime* Date for which a history file should be opened.

*suffix* File name suffix like "hst", "idx", "idf"

*mode* R/W access mode

*fh* File handle

##### Returns:

HS\_SUCCESS

Definition at line 12966 of file midas.c.

#### 6.34.1.2 INT **hs\_set\_path** (char \* *path*)

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

##### Parameters:

*path* Directory where history files reside

##### Returns:

HS\_SUCCESS

Definition at line 12940 of file midas.c.

## 6.35 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 \*filename1, char \*buffer1, INT buffer\_size1, char \*filename2, char \*buffer2, INT buffer\_size2, char \*filename3, char \*buffer3, INT buffer\_size3, char \*tag, INT tag\_size)

### 6.35.1 Function Documentation

**6.35.1.1** **INT el\_submit** (int *run*, char \* *author*, char \* *type*, char \* *system*, char \* *subject*, char \* *text*, char \* *reply\_to*, char \* *encoding*, char \* *filename1*, char \* *buffer1*, INT *buffer\_size1*, char \* *filename2*, char \* *buffer2*, INT *buffer\_size2*, char \* *filename3*, 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.  
*filename1* File name of attachment.  
*buffer1* File contents.  
*buffer\_size1* Size of buffer in bytes.  
*filename2* File name of attachment.  
*buffer2* File contents.  
*buffer\_size2* Size of buffer in bytes.  
*filename3* File name of attachment.  
*buffer3* File contents.  
*buffer\_size3* Size of buffer in bytes.

*tag* If given, edit existing message.

*tag\_size* Maximum size of tag.

**Returns:**

EL\_SUCCESS

Definition at line 14763 of file midas.c.

## 6.36 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** (HANDLE hConn, INT item, INT value)
- **INT rpc\_send\_event** (INT buffer\_handle, void \*source, INT buf\_size, INT async\_flag)
- **INT rpc\_flush\_event** ()

### 6.36.1 Function Documentation

#### 6.36.1.1 INT rpc\_flush\_event ()

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

**Returns:**

RPC\_SUCCESS, RPC\_NET\_ERROR

Definition at line 10057 of file midas.c.

Referenced by main(), scheduler(), and send\_event().

#### 6.36.1.2 INT rpc\_register\_client (char \* name, RPC\_LIST \* list)

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

**Parameters:**

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

**name** Name of this client

**Returns:**

RPC\_SUCCESS

Definition at line 8034 of file midas.c.

### 6.36.1.3 INT rpc\_register\_functions (RPC\_LIST \* *new\_list*, INT(\* *func*)(INT, void \*\*))

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

#### Parameters:

*new\_list* Array of RPC\_LIST structures containing function IDs and parameter definitions. The end of the list must be indicated by a function ID of zero.

*func* Default dispatch function

#### Returns:

RPC\_SUCCESS, RPC\_NO\_MEMORY, RPC\_DOUBLE\_DEFINED

Definition at line 8054 of file midas.c.

Referenced by cm\_connect\_experiment1(), and rpc\_register\_client().

### 6.36.1.4 INT rpc\_send\_event (INT *buffer\_handle*, void \* *source*, INT *buf\_size*, INT *async\_flag*)

Fast send\_event routine which bypasses the RPC layer and sends the event directly at the TCP level.

#### Parameters:

*buffer\_handle* Handle of the buffer to send the event to. Must be obtained via bm\_open\_buffer.

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

*buf\_size* Size of event in bytes with header.

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

#### Returns:

BM\_INVALID\_PARAM, BM\_ASYNC\_RETURN, RPC\_SUCCESS, RPC\_NET\_ERROR, RPC\_NO\_CONNECTION, RPC\_EXCEED\_BUFFER

Definition at line 9867 of file midas.c.

Referenced by interrupt\_routine(), scheduler(), and source\_scan().

**6.36.1.5 INT `rpc_set_option` (HANDLE *hConn*, INT *item*, INT *value*)**

Set RPC option

**Parameters:**

*hConn* RPC connection handle

*item* One of RPC\_Oxxx

*value* Value to set

**Returns:**

RPC\_SUCCESS

Definition at line 9045 of file midas.c.

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

## 6.37 Midas Dual Buffer Memory Functions (dm\_XXX)

### Functions

- INT **dm\_buffer\_create** (INT *size*, INT *user\_max\_event\_size*)

#### 6.37.1 Function Documentation

##### 6.37.1.1 INT **dm\_buffer\_create** (INT *size*, INT *user\_max\_event\_size*)

Setup a dual memory buffer. Has to be called initially before any other dm\_XXX function

#### Parameters:

*size* Size in bytes

*user\_max\_event\_size* max event size

#### Returns:

CM\_SUCCESS, BM\_NO\_MEMORY, BM\_MEMSIZE\_MISMATCH

Definition at line 16736 of file midas.c.

Referenced by `main()`.

## 6.38 System Functions (ss\_XXX)

### Functions

- INT **ss\_thread\_create** (INT(\*thread\_func)(void \*), void \*param)
- INT **ss\_thread\_kill** (INT thread\_id)
- DWORD **ss\_millitime** ()
- DWORD **ss\_time** ()
- INT **ss\_sleep** (INT millisec)

### 6.38.1 Function Documentation

#### 6.38.1.1 DWORD ss\_millitime ()

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

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

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

#### Returns:

millisecond time stamp.

Definition at line 2253 of file system.c.

Referenced by `bm_check_buffers()`, `bm_open_buffer()`, `cm_cleanup()`, `cm_get_watchdog_info()`, `cm_set_watchdog_params()`, `cm_shutdown()`, `db_open_database()`, `dm_buffer_create()`, `main()`, `scheduler()`, `send_event()`, `tr_prestart()`, and `tr_stop()`.

#### 6.38.1.2 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()`(p.??)



**Parameters:**

*millisec* Time in milliseconds to sleep. Zero means infinite (until another process calls ss\_wake)

**Returns:**

SS\_SUCCESS

Definition at line 2447 of file system.c.

Referenced by cm\_shutdown(), and main().

### 6.38.1.3 INT ss\_thread\_create (INT(\* thread\_func)(void \*), void \* param)

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

```
{ ...
  char cmd[256];
  sprintf(cmd,"%s %s %i %s/%s %1.3lf %d",lazy.commandAfter,
    lazy.backlabel, lazyst.nfiles, lazy.path, lazyst.backfile,
    lazyst.file_size/1024.0/1024.0, blockn);
  cm_msg(MINFO,"Lazy","Exec post file write script:%s",cmd);
  ss_system(cmd);
}
...
\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};
thread_id = ss_thread_spawn((void *) taskWatch, &tsWatch);
if (thread_id == 0) {
    printf("cannot spawn taskWatch\n");
}
...

```

Example for Linux

```

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

```

#### Parameters:

(*\*thread\_func*) Thread function to create.

*param* a pointer to a VX\_TASK\_SPAWN structure for vxWorks and a void pointer for Unix and Windows

#### Returns:

the new thread id or zero on error

Definition at line 1676 of file system.c.

Referenced by dm\_buffer\_create().

**6.38.1.4 INT ss\_thread\_kill (INT *thread\_id*)**

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

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

**Parameters:**

*thread\_id* the thread id of the thread to be killed.

**Returns:**

SS\_SUCCESS if no error, else SS\_NO\_THREAD

Definition at line 1753 of file system.c.

**6.38.1.5 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

Definition at line 2320 of file 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()`.

## 6.39 The msystem.h & system.c

### Modules

- System Functions (ss\_\*)
- System #define
- System Macros
- System Structure Declaration

## 6.40 System #define

### Defines

- #define **DRI\_16** (1<<0)
- #define **DRI\_32** (1<<1)
- #define **DRI\_64** (1<<2)
- #define **DRI\_LITTLE\_ENDIAN** (1<<3)
- #define **DRI\_BIG\_ENDIAN** (1<<4)
- #define **DRF\_IEEE** (1<<5)
- #define **DRF\_G\_FLOAT** (1<<6)
- #define **DR\_ASCII** (1<<7)

### 6.40.1 Define Documentation

#### 6.40.1.1 #define DR\_ASCII (1<<7)

- 

Definition at line 161 of file msystem.h.

#### 6.40.1.2 #define DRF\_G\_FLOAT (1<<6)

- 

Definition at line 160 of file msystem.h.

#### 6.40.1.3 #define DRF\_IEEE (1<<5)

- 

Definition at line 159 of file msystem.h.

#### 6.40.1.4 #define DRI\_16 (1<<0)

- 

Definition at line 154 of file msystem.h.

**6.40.1.5** `#define DRI_32 (1<<1)`

- 

Definition at line 155 of file `msystem.h`.

**6.40.1.6** `#define DRI_64 (1<<2)`

- 

Definition at line 156 of file `msystem.h`.

**6.40.1.7** `#define DRI_BIG_ENDIAN (1<<4)`

- 

Definition at line 158 of file `msystem.h`.

**6.40.1.8** `#define DRI_LITTLE_ENDIAN (1<<3)`

- 

Definition at line 157 of file `msystem.h`.

## 6.41 System Macros

### Defines

- #define **WORD\_SWAP**(x)
- #define **DWORD\_SWAP**(x)
- #define **QWORD\_SWAP**(x)

#### 6.41.1 Define Documentation

##### 6.41.1.1 #define **DWORD\_SWAP**(x)

###### Value:

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

SWAP DWORD macro

Definition at line 181 of file msystem.h.

Referenced by bk\_swap().

##### 6.41.1.2 #define **QWORD\_SWAP**(x)

###### Value:

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

SWAP QWORD macro

Definition at line 191 of file msystem.h.

Referenced by bk\_swap().



### 6.41.1.3 `#define WORD_SWAP(x)`

**Value:**

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

SWAP WORD macro

Definition at line 174 of file msystem.h.

Referenced by `bk_swap()`.

## 6.42 System Structure Declaration

### Data Structures

- struct **DATABASE**
- struct **DATABASE\_CLIENT**
- struct **DATABASE\_HEADER**
- struct **FREE\_DESCRIP**
- struct **OPEN\_RECORD**
- struct **RECORD\_LIST**
- struct **REQUEST\_LIST**

## 6.43 The mrpc.h & mrpc.c

### Modules

- `RPC #define`
- `Midas RPC_LIST`

## 6.44 RPC #define

### Defines

- #define **RPC\_CM\_SET\_CLIENT\_INFO** 11000
- #define **RPC\_CM\_SET\_WATCHDOG\_PARAMS** 11001
- #define **RPC\_CM\_CLEANUP** 11002
- #define **RPC\_CM\_GET\_WATCHDOG\_INFO** 11003
- #define **RPC\_CM\_MSG\_LOG** 11004
- #define **RPC\_CM\_EXECUTE** 11005
- #define **RPC\_CM\_SYNCHRONIZE** 11006
- #define **RPC\_CM\_ASCTIME** 11007
- #define **RPC\_CM\_TIME** 11008
- #define **RPC\_CM\_MSG** 11009
- #define **RPC\_CM\_EXIST** 11011
- #define **RPC\_CM\_MSG\_RETRIEVE** 11012
- #define **RPC\_CM\_MSG\_LOG1** 11013
- #define **RPC\_BM\_OPEN\_BUFFER** 11100
- #define **RPC\_BM\_CLOSE\_BUFFER** 11101
- #define **RPC\_BM\_CLOSE\_ALL\_BUFFERS** 11102
- #define **RPC\_BM\_GET\_BUFFER\_INFO** 11103
- #define **RPC\_BM\_GET\_BUFFER\_LEVEL** 11104
- #define **RPC\_BM\_INIT\_BUFFER\_COUNTERS** 11105
- #define **RPC\_BM\_SET\_CACHE\_SIZE** 11106
- #define **RPC\_BM\_ADD\_EVENT\_REQUEST** 11107
- #define **RPC\_BM\_REMOVE\_EVENT\_REQUEST** 11108
- #define **RPC\_BM\_SEND\_EVENT** 11109
- #define **RPC\_BM\_FLUSH\_CACHE** 11110
- #define **RPC\_BM\_RECEIVE\_EVENT** 11111
- #define **RPC\_BM\_MARK\_READ\_WAITING** 11112
- #define **RPC\_BM\_EMPTY\_BUFFERS** 11113
- #define **RPC\_BM\_SKIP\_EVENT** 11114
- #define **RPC\_DB\_OPEN\_DATABASE** 11200
- #define **RPC\_DB\_CLOSE\_DATABASE** 11201
- #define **RPC\_DB\_CLOSE\_ALL\_DATABASES** 11202
- #define **RPC\_DB\_CREATE\_KEY** 11203
- #define **RPC\_DB\_CREATE\_LINK** 11204
- #define **RPC\_DB\_SET\_VALUE** 11205
- #define **RPC\_DB\_GET\_VALUE** 11206
- #define **RPC\_DB\_FIND\_KEY** 11207
- #define **RPC\_DB\_FIND\_LINK** 11208
- #define **RPC\_DB\_GET\_PATH** 11209
- #define **RPC\_DB\_DELETE\_KEY** 11210

- #define **RPC\_DB\_ENUM\_KEY** 11211
- #define **RPC\_DB\_GET\_KEY** 11212
- #define **RPC\_DB\_GET\_DATA** 11213
- #define **RPC\_DB\_SET\_DATA** 11214
- #define **RPC\_DB\_SET\_DATA\_INDEX** 11215
- #define **RPC\_DB\_SET\_MODE** 11216
- #define **RPC\_DB\_GET\_RECORD\_SIZE** 11219
- #define **RPC\_DB\_GET\_RECORD** 11220
- #define **RPC\_DB\_SET\_RECORD** 11221
- #define **RPC\_DB\_ADD\_OPEN\_RECORD** 11222
- #define **RPC\_DB\_REMOVE\_OPEN\_RECORD** 11223
- #define **RPC\_DB\_SAVE** 11224
- #define **RPC\_DB\_LOAD** 11225
- #define **RPC\_DB\_SET\_CLIENT\_NAME** 11226
- #define **RPC\_DB\_RENAME\_KEY** 11227
- #define **RPC\_DB\_ENUM\_LINK** 11228
- #define **RPC\_DB\_REORDER\_KEY** 11229
- #define **RPC\_DB\_CREATE\_RECORD** 11230
- #define **RPC\_DB\_GET\_DATA\_INDEX** 11231
- #define **RPC\_DB\_GET\_KEY\_TIME** 11232
- #define **RPC\_DB\_GET\_OPEN\_RECORDS** 11233
- #define **RPC\_DB\_FLUSH\_DATABASE** 11235
- #define **RPC\_DB\_SET\_DATA\_INDEX2** 11236
- #define **RPC\_DB\_GET\_KEY\_INFO** 11237
- #define **RPC\_DB\_GET\_DATA1** 11238
- #define **RPC\_DB\_SET\_NUM\_VALUES** 11239
- #define **RPC\_DB\_CHECK\_RECORD** 11240
- #define **RPC\_DB\_GET\_NEXT\_LINK** 11241
- #define **RPC\_HS\_SET\_PATH** 11300
- #define **RPC\_HS\_DEFINE\_EVENT** 11301
- #define **RPC\_HS\_WRITE\_EVENT** 11302
- #define **RPC\_HS\_COUNT\_EVENTS** 11303
- #define **RPC\_HS\_ENUM\_EVENTS** 11304
- #define **RPC\_HS\_COUNT\_VARS** 11305
- #define **RPC\_HS\_ENUM\_VARS** 11306
- #define **RPC\_HS\_READ** 11307
- #define **RPC\_HS\_GET\_VAR** 11308
- #define **RPC\_HS\_GET\_EVENT\_ID** 11309
- #define **RPC\_EL\_SUBMIT** 11400
- #define **RPC\_AL\_CHECK** 11500
- #define **RPC\_AL\_TRIGGER\_ALARM** 11501
- #define **RPC\_RC\_TRANSITION** 12000
- #define **RPC\_ANA\_CLEAR\_HISTOS** 13000

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

### 6.44.1 Define Documentation

#### 6.44.1.1 `#define RPC_AL_CHECK 11500`

- 

Definition at line 168 of file `mrpc.h`.

#### 6.44.1.2 `#define RPC_AL_TRIGGER_ALARM 11501`

- 

Definition at line 169 of file `mrpc.h`.

Referenced by `al_trigger_alarm()`.

#### 6.44.1.3 `#define RPC_ANA_CLEAR_HISTOS 13000`

- 

Definition at line 173 of file `mrpc.h`.

#### 6.44.1.4 `#define RPC_BM_ADD_EVENT_REQUEST 11107`

- 

Definition at line 106 of file `mrpc.h`.

#### 6.44.1.5 `#define RPC_BM_CLOSE_ALL_BUFFERS 11102`

- 

Definition at line 101 of file `mrpc.h`.

Referenced by `bm_close_all_buffers()`.

**6.44.1.6 #define RPC\_BM\_CLOSE\_BUFFER 11101**

- 

Definition at line 100 of file mrpc.h.

Referenced by bm\_close\_buffer().

**6.44.1.7 #define RPC\_BM\_EMPTY\_BUFFERS 11113**

- 

Definition at line 112 of file mrpc.h.

Referenced by bm\_empty\_buffers().

**6.44.1.8 #define RPC\_BM\_FLUSH\_CACHE 11110**

- 

Definition at line 109 of file mrpc.h.

Referenced by bm\_flush\_cache().

**6.44.1.9 #define RPC\_BM\_GET\_BUFFER\_INFO 11103**

- 

Definition at line 102 of file mrpc.h.

**6.44.1.10 #define RPC\_BM\_GET\_BUFFER\_LEVEL 11104**

- 

Definition at line 103 of file mrpc.h.

**6.44.1.11 #define RPC\_BM\_INIT\_BUFFER\_COUNTERS  
11105**

- 

Definition at line 104 of file mrpc.h.

**6.44.1.12   #define RPC\_BM\_MARK\_READ\_WAITING 11112**

- 

Definition at line 111 of file mrpc.h.

**6.44.1.13   #define RPC\_BM\_OPEN\_BUFFER 11100**

- 

Definition at line 99 of file mrpc.h.

Referenced by bm\_open\_buffer().

**6.44.1.14   #define RPC\_BM\_RECEIVE\_EVENT 11111**

- 

Definition at line 110 of file mrpc.h.

Referenced by bm\_receive\_event().

**6.44.1.15   #define RPC\_BM\_REMOVE\_EVENT\_REQUEST  
11108**

- 

Definition at line 107 of file mrpc.h.

Referenced by bm\_remove\_event\_request().

**6.44.1.16   #define RPC\_BM\_SEND\_EVENT 11109**

- 

Definition at line 108 of file mrpc.h.

Referenced by bm\_send\_event(), and rpc\_send\_event().

**6.44.1.17   #define RPC\_BM\_SET\_CACHE\_SIZE 11106**

- 

Definition at line 105 of file mrpc.h.

Referenced by bm\_set\_cache\_size().



**6.44.1.18 #define RPC\_BM\_SKIP\_EVENT 11114**

- 

Definition at line 113 of file mrpc.h.

Referenced by bm\_skip\_event().

**6.44.1.19 #define RPC\_CM\_ASCTIME 11007**

- 

Definition at line 92 of file mrpc.h.

Referenced by cm\_asctime().

**6.44.1.20 #define RPC\_CM\_CLEANUP 11002**

- 

Definition at line 87 of file mrpc.h.

Referenced by cm\_cleanup().

**6.44.1.21 #define RPC\_CM\_EXECUTE 11005**

- 

Definition at line 90 of file mrpc.h.

Referenced by cm\_execute().

**6.44.1.22 #define RPC\_CM\_EXIST 11011**

- 

Definition at line 95 of file mrpc.h.

Referenced by cm\_exist().

**6.44.1.23 #define RPC\_CM\_GET\_WATCHDOG\_INFO 11003**

-

Definition at line 88 of file mrpc.h.

Referenced by `cm_get_watchdog_info()`.

**6.44.1.24    `#define RPC_CM_MSG 11009`**

- 

Definition at line 94 of file mrpc.h.

**6.44.1.25    `#define RPC_CM_MSG_LOG 11004`**

- 

Definition at line 89 of file mrpc.h.

Referenced by `cm_msg_log()`.

**6.44.1.26    `#define RPC_CM_MSG_LOG1 11013`**

- 

Definition at line 97 of file mrpc.h.

Referenced by `cm_msg_log1()`.

**6.44.1.27    `#define RPC_CM_MSG_RETRIEVE 11012`**

- 

Definition at line 96 of file mrpc.h.

Referenced by `cm_msg_retrieve()`.

**6.44.1.28    `#define RPC_CM_SET_CLIENT_INFO 11000`**

- 

Definition at line 85 of file mrpc.h.

Referenced by `cm_set_client_info()`.

**6.44.1.29 #define RPC\_CM\_SET\_WATCHDOG\_PARAMS 11001**

- 

Definition at line 86 of file mrpc.h.

Referenced by cm\_set\_watchdog\_params().

**6.44.1.30 #define RPC\_CM\_SYNCHRONIZE 11006**

- 

Definition at line 91 of file mrpc.h.

Referenced by cm\_synchronize().

**6.44.1.31 #define RPC\_CM\_TIME 11008**

- 

Definition at line 93 of file mrpc.h.

Referenced by cm\_time().

**6.44.1.32 #define RPC\_CNAF16 16000**

- 

Definition at line 179 of file mrpc.h.

Referenced by main().

**6.44.1.33 #define RPC\_CNAF24 16001**

- 

Definition at line 180 of file mrpc.h.

Referenced by main().

**6.44.1.34 #define RPC\_DB\_ADD\_OPEN\_RECORD 11222**

- 

Definition at line 135 of file mrpc.h.

**6.44.1.35   #define RPC\_DB\_CHECK\_RECORD 11240**

- 

Definition at line 152 of file mrpc.h.

Referenced by db\_check\_record().

**6.44.1.36   #define RPC\_DB\_CLOSE\_ALL\_DATABASES 11202**

- 

Definition at line 117 of file mrpc.h.

**6.44.1.37   #define RPC\_DB\_CLOSE\_DATABASE 11201**

- 

Definition at line 116 of file mrpc.h.

Referenced by db\_close\_database().

**6.44.1.38   #define RPC\_DB\_CREATE\_KEY 11203**

- 

Definition at line 118 of file mrpc.h.

Referenced by db\_create\_key().

**6.44.1.39   #define RPC\_DB\_CREATE\_LINK 11204**

- 

Definition at line 119 of file mrpc.h.

Referenced by db\_create\_link().

**6.44.1.40   #define RPC\_DB\_CREATE\_RECORD 11230**

- 

Definition at line 143 of file mrpc.h.

Referenced by db\_create\_record().

**6.44.1.41 #define RPC\_DB\_DELETE\_KEY 11210**

- 

Definition at line 125 of file mrpc.h.

Referenced by db\_delete\_key().

**6.44.1.42 #define RPC\_DB\_ENUM\_KEY 11211**

- 

Definition at line 126 of file mrpc.h.

Referenced by db\_enum\_key().

**6.44.1.43 #define RPC\_DB\_ENUM\_LINK 11228**

- 

Definition at line 141 of file mrpc.h.

**6.44.1.44 #define RPC\_DB\_FIND\_KEY 11207**

- 

Definition at line 122 of file mrpc.h.

Referenced by db\_find\_key().

**6.44.1.45 #define RPC\_DB\_FIND\_LINK 11208**

- 

Definition at line 123 of file mrpc.h.

**6.44.1.46 #define RPC\_DB\_FLUSH\_DATABASE 11235**

- 

Definition at line 147 of file mrpc.h.

**6.44.1.47 #define RPC\_DB\_GET\_DATA 11213**

- 

Definition at line 128 of file mrpc.h.

Referenced by db\_get\_data().

**6.44.1.48 #define RPC\_DB\_GET\_DATA1 11238**

- 

Definition at line 150 of file mrpc.h.

**6.44.1.49 #define RPC\_DB\_GET\_DATA\_INDEX 11231**

- 

Definition at line 144 of file mrpc.h.

Referenced by db\_get\_data\_index().

**6.44.1.50 #define RPC\_DB\_GET\_KEY 11212**

- 

Definition at line 127 of file mrpc.h.

Referenced by db\_get\_key().

**6.44.1.51 #define RPC\_DB\_GET\_KEY\_INFO 11237**

- 

Definition at line 149 of file mrpc.h.

Referenced by db\_get\_key\_info().

**6.44.1.52 #define RPC\_DB\_GET\_KEY\_TIME 11232**

- 

Definition at line 145 of file mrpc.h.

Referenced by db\_get\_key\_time().

**6.44.1.53 #define RPC\_DB\_GET\_NEXT\_LINK 11241**

- 

Definition at line 153 of file mrpc.h.

**6.44.1.54 #define RPC\_DB\_GET\_OPEN\_RECORDS 11233**

- 

Definition at line 146 of file mrpc.h.

**6.44.1.55 #define RPC\_DB\_GET\_PATH 11209**

- 

Definition at line 124 of file mrpc.h.

**6.44.1.56 #define RPC\_DB\_GET\_RECORD 11220**

- 

Definition at line 133 of file mrpc.h.

Referenced by db\_get\_record().

**6.44.1.57 #define RPC\_DB\_GET\_RECORD\_SIZE 11219**

- 

Definition at line 132 of file mrpc.h.

Referenced by db\_get\_record\_size().

**6.44.1.58 #define RPC\_DB\_GET\_VALUE 11206**

- 

Definition at line 121 of file mrpc.h.

Referenced by db\_get\_value().

**6.44.1.59    #define RPC\_DB\_LOAD 11225**

- 

Definition at line 138 of file mrpc.h.

Referenced by db\_load().

**6.44.1.60    #define RPC\_DB\_OPEN\_DATABASE 11200**

- 

Definition at line 115 of file mrpc.h.

Referenced by db\_open\_database().

**6.44.1.61    #define RPC\_DB\_REMOVE\_OPEN\_RECORD 11223**

- 

Definition at line 136 of file mrpc.h.

**6.44.1.62    #define RPC\_DB\_RENAME\_KEY 11227**

- 

Definition at line 140 of file mrpc.h.

**6.44.1.63    #define RPC\_DB\_REORDER\_KEY 11229**

- 

Definition at line 142 of file mrpc.h.

**6.44.1.64    #define RPC\_DB\_SAVE 11224**

- 

Definition at line 137 of file mrpc.h.

Referenced by db\_save().



**6.44.1.65 #define RPC\_DB\_SET\_CLIENT\_NAME 11226**

- 

Definition at line 139 of file mrpc.h.

**6.44.1.66 #define RPC\_DB\_SET\_DATA 11214**

- 

Definition at line 129 of file mrpc.h.

Referenced by db\_set\_data().

**6.44.1.67 #define RPC\_DB\_SET\_DATA\_INDEX 11215**

- 

Definition at line 130 of file mrpc.h.

Referenced by db\_set\_data\_index().

**6.44.1.68 #define RPC\_DB\_SET\_DATA\_INDEX2 11236**

- 

Definition at line 148 of file mrpc.h.

**6.44.1.69 #define RPC\_DB\_SET\_MODE 11216**

- 

Definition at line 131 of file mrpc.h.

**6.44.1.70 #define RPC\_DB\_SET\_NUM\_VALUES 11239**

- 

Definition at line 151 of file mrpc.h.

**6.44.1.71 #define RPC\_DB\_SET\_RECORD 11221**

- 

Definition at line 134 of file mrpc.h.

Referenced by db\_set\_record().

**6.44.1.72 #define RPC\_DB\_SET\_VALUE 11205**

- 

Definition at line 120 of file mrpc.h.

Referenced by db\_set\_value().

**6.44.1.73 #define RPC\_EL\_SUBMIT 11400**

- 

Definition at line 166 of file mrpc.h.

Referenced by el\_submit().

**6.44.1.74 #define RPC\_HS\_COUNT\_EVENTS 11303**

- 

Definition at line 158 of file mrpc.h.

**6.44.1.75 #define RPC\_HS\_COUNT\_VARS 11305**

- 

Definition at line 160 of file mrpc.h.

**6.44.1.76 #define RPC\_HS\_DEFINE\_EVENT 11301**

- 

Definition at line 156 of file mrpc.h.

**6.44.1.77 #define RPC\_HS\_ENUM\_EVENTS 11304**

- 

Definition at line 159 of file mrpc.h.

**6.44.1.78 #define RPC\_HS\_ENUM\_VARS 11306**

- 

Definition at line 161 of file mrpc.h.

**6.44.1.79 #define RPC\_HS\_GET\_EVENT\_ID 11309**

- 

Definition at line 164 of file mrpc.h.

**6.44.1.80 #define RPC\_HS\_GET\_VAR 11308**

- 

Definition at line 163 of file mrpc.h.

**6.44.1.81 #define RPC\_HS\_READ 11307**

- 

Definition at line 162 of file mrpc.h.

**6.44.1.82 #define RPC\_HS\_SET\_PATH 11300**

- 

Definition at line 155 of file mrpc.h.

Referenced by `hs_set_path()`.

**6.44.1.83 #define RPC\_HS\_WRITE\_EVENT 11302**

- 

Definition at line 157 of file mrpc.h.

**6.44.1.84    #define RPC\_ID\_EXIT 99999**

- 

Definition at line 186 of file mrpc.h.

**6.44.1.85    #define RPC\_ID\_SHUTDOWN 99998**

- 

Definition at line 185 of file mrpc.h.

**6.44.1.86    #define RPC\_ID\_WATCHDOG 99997**

- 

Definition at line 184 of file mrpc.h.

**6.44.1.87    #define RPC\_LOG\_REWIND 14000**

- 

Definition at line 175 of file mrpc.h.

**6.44.1.88    #define RPC\_MANUAL\_TRIG 17000**

- 

Definition at line 182 of file mrpc.h.

**6.44.1.89    #define RPC\_RC\_TRANSITION 12000**

- 

Definition at line 171 of file mrpc.h.

Referenced by cm\_register\_transition(), and cm\_transition().

**6.44.1.90    #define RPC\_TEST 15000**

- 

Definition at line 177 of file mrpc.h.

## 6.45 Midas RPC\_LIST

### Variables

- RPC\_LIST `rpc_list_library` []
- RPC\_LIST `rpc_list_system` []

### 6.45.1 Function Documentation

#### 6.45.1.1 RPC\_LIST\* `rpc_get_internal_list` (INT *flag*)

Definition at line 1338 of file `mrpc.c`.

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

### 6.45.2 Variable Documentation

#### 6.45.2.1 RPC\_LIST `rpc_list_library`[] [static]

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

Definition at line 127 of file `mrpc.c`.

Referenced by `rpc_get_internal_list()`.

#### 6.45.2.2 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 registered whenever a RPC server is registered

Definition at line 1321 of file `mrpc.c`.

Referenced by `rpc_get_internal_list()`.

## 6.46 The odb.c

### Modules

- Midas ODB Functions (db\_XXX)

## 6.47 Midas ODB Functions (db\_XXX)

### Functions

- **INT db\_open\_database** (char \*database\_name, INT database\_size, HANDLE \*hDB, char \*client\_name)
- **INT db\_close\_database** (HANDLE hDB)
- **INT db\_lock\_database** (HANDLE hDB)
- **INT db\_unlock\_database** (HANDLE hDB)
- **INT db\_protect\_database** (HANDLE hDB)
- **INT db\_create\_key** (HANDLE hDB, HANDLE hKey, char \*key\_name, DWORD type)
- **INT db\_create\_link** (HANDLE hDB, HANDLE hKey, char \*link\_name, char \*destination)
- **INT db\_delete\_key1** (HANDLE hDB, HANDLE hKey, INT level, BOOL follow\_links)
- **INT db\_delete\_key** (HANDLE hDB, HANDLE hKey, BOOL follow\_links)
- **INT db\_find\_key** (HANDLE hDB, HANDLE hKey, char \*key\_name, HANDLE \*subhKey)
- **INT db\_set\_value** (HANDLE hDB, HANDLE hKeyRoot, char \*key\_name, void \*data, INT data\_size, INT num\_values, DWORD type)
- **INT db\_get\_value** (HANDLE hDB, HANDLE hKeyRoot, char \*key\_name, void \*data, INT \*buf\_size, DWORD type, BOOL create)
- **INT db\_enum\_key** (HANDLE hDB, HANDLE hKey, INT index, HANDLE \*subkey\_handle)
- **INT db\_get\_key** (HANDLE hDB, HANDLE hKey, KEY \*key)
- **INT db\_get\_key\_time** (HANDLE hDB, HANDLE hKey, DWORD \*delta)
- **INT db\_get\_key\_info** (HANDLE hDB, HANDLE hKey, char \*name, INT name\_size, INT \*type, INT \*num\_values, INT \*item\_size)
- **INT db\_get\_data** (HANDLE hDB, HANDLE hKey, void \*data, INT \*buf\_size, DWORD type)
- **INT db\_get\_data\_index** (HANDLE hDB, HANDLE hKey, void \*data, INT \*buf\_size, INT index, DWORD type)
- **INT db\_set\_data** (HANDLE hDB, HANDLE hKey, void \*data, INT buf\_size, INT num\_values, DWORD type)
- **INT db\_set\_data\_index** (HANDLE hDB, HANDLE hKey, void \*data, INT data\_size, INT index, DWORD type)
- **INT db\_load** (HANDLE hDB, HANDLE hKeyRoot, char \*filename, BOOL bRemote)
- **INT db\_copy** (HANDLE hDB, HANDLE hKey, char \*buffer, INT \*buffer\_size, char \*path)
- **INT db\_paste** (HANDLE hDB, HANDLE hKeyRoot, char \*buffer)



- INT **db\_save** (HANDLE **hDB**, HANDLE **hKey**, char \*filename, BOOL bRemote)
- INT **db\_save\_struct** (HANDLE **hDB**, HANDLE **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** (HANDLE **hDB**, HANDLE **hKey**, INT align, INT \*buf\_size)
- INT **db\_get\_record** (HANDLE **hDB**, HANDLE **hKey**, void \*data, INT \*buf\_size, INT align)
- INT **db\_set\_record** (HANDLE **hDB**, HANDLE **hKey**, void \*data, INT buf\_size, INT align)
- INT **db\_create\_record** (HANDLE **hDB**, HANDLE **hKey**, char \*orig\_key\_name, char \*init\_str)
- INT **db\_check\_record** (HANDLE **hDB**, HANDLE **hKey**, char \*keyname, char \*rec\_str, BOOL correct)
- INT **db\_open\_record** (HANDLE **hDB**, HANDLE **hKey**, void \*ptr, INT rec\_size, WORD access\_mode, void(\*dispatcher)(INT, INT, void \*), void \*info)
- INT **db\_close\_record** (HANDLE **hDB**, HANDLE **hKey**)
- INT **db\_close\_all\_records** ()
- INT **db\_update\_record** (INT **hDB**, INT **hKey**, int socket)
- INT **db\_send\_changed\_records** ()

### 6.47.1 Function Documentation

#### 6.47.1.1 INT **db\_check\_record** (HANDLE **hDB**, HANDLE **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**(p.??) if **correct**=TRUE.

#### Parameters:

**hDB** ODB handle obtained via **cm\_get\_experiment\_database**(p.??).

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

**keyname** Name of key to search, can contain directories.

**rec\_str** ASCII representation of ODB record in the format

**correct** If TRUE, correct ODB record if necessary

**Returns:**

DB\_SUCCESS, DB\_INVALID\_HANDLE, DB\_NO\_KEY, DB\_STRUCT\_MISMATCH

Definition at line 7227 of file odb.c.

Referenced by al\_trigger\_alarm(), and cm\_connect\_experiment1().

**6.47.1.2 INT db\_close\_all\_records ()**

Release local memory for open records. This routines is called by db\_close\_all\_databases() and **cm\_disconnect\_experiment()**(p.??)

**Returns:**

DB\_SUCCESS, DB\_INVALID\_HANDLE

Definition at line 7709 of file odb.c.

Referenced by cm\_disconnect\_experiment().

**6.47.1.3 INT db\_close\_database (HANDLE *hDB*)**

Close a database

**Parameters:**

*hDB* ODB handle obtained via **cm\_get\_experiment\_database()**(p.??).

**Returns:**

DB\_SUCCESS, DB\_INVALID\_HANDLE, RPC\_NET\_ERROR

Definition at line 1274 of file odb.c.

**6.47.1.4 INT db\_close\_record (HANDLE *hDB*, HANDLE *hKey*)**

Close a record previously opened with db\_open\_record.

**Parameters:**

*hDB* ODB handle obtained via **cm\_get\_experiment\_database()**(p.??).

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

**Returns:**

DB\_SUCCESS, DB\_INVALID\_HANDLE

Definition at line 7672 of file odb.c.

#### 6.47.1.5 INT db\_copy (HANDLE *hDB*, HANDLE *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()**(p.??) can be used to convert the ASCII representation back to binary ODB contents. The functions **db\_load()**(p.??) and **db\_save()**(p.??) internally use **db\_copy()**(p.??) and **db\_paste()**(p.??). 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()**(p.??).

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

**buffer** ASCII buffer which receives ODB contents.

**buffer\_size** Size of buffer, returns remaining space in buffer.

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

#### Returns:

DB\_SUCCESS, DB\_TRUNCATED, DB\_NO\_MEMORY

Definition at line 5195 of file odb.c.

Referenced by **db\_create\_record()**, and **db\_save()**.

#### 6.47.1.6 INT db\_create\_key (HANDLE *hDB*, HANDLE *hKey*, char \* *key\_name*, DWORD *type*)

Create a new key in a database

##### Parameters:

*hDB* ODB handle obtained via **cm\_get\_experiment\_database()**(p. ??).

*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**(p. ??))

##### Returns:

DB\_SUCCESS, DB\_INVALID\_HANDLE, DB\_FULL, DB\_KEY\_EXIST, DB\_NO\_ACCESS

Definition at line 1678 of file odb.c.

Referenced by db\_create\_record(), db\_get\_value(), db\_paste(), and db\_set\_value().

#### 6.47.1.7 INT db\_create\_link (HANDLE *hDB*, HANDLE *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()**(p. ??).

*hKey* Key handle to start with, 0 for root

*link\_name* Name of key in the form "/key/key/key"

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

##### Returns:

DB\_SUCCESS, DB\_INVALID\_HANDLE, DB\_FULL, DB\_KEY\_EXIST, DB\_NO\_ACCESS

Definition at line 1913 of file odb.c.

#### 6.47.1.8 INT db\_create\_record (HANDLE *hDB*, HANDLE *hKey*, char \* *orig\_key\_name*, char \* *init\_str*)

Create a record. If a part of the record exists already, 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()**(p.??) 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()**(p.??).

If a sub-tree exists already which exactly matches the ASCII representation, it is not modified. If part of the tree exists, it is merged with the ASCII representation where the ODB values have priority, only values not present in the ODB are created with the default values of the ASCII representation. It is therefore recommended that before creating an ODB hot-link the function **db\_create\_record()**(p.??) 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()
{
    HANDLE 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()**(p.??).

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

**orig\_key\_name** Name of key to search, can contain directories.

*init\_str* Initialization string in the format of the db\_copy/db\_save functions.

**Returns:**

DB\_SUCCESS, DB\_INVALID\_HANDLE, DB\_FULL, DB\_NO\_ACCESS, DB\_OPEN\_RECORD

Definition at line 7055 of file odb.c.

Referenced by al\_trigger\_alarm(), cm\_set\_client\_info(), db\_check\_record(), and main().

#### 6.47.1.9 INT db\_delete\_key (HANDLE *hDB*, HANDLE *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()(p. ??).

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

*follow\_links* Follow links when TRUE.

**Returns:**

DB\_SUCCESS, DB\_INVALID\_HANDLE, DB\_NO\_ACCESS, DB\_OPEN\_RECORD

Definition at line 2116 of file odb.c.

Referenced by cm\_set\_client\_info(), and db\_create\_record().

#### 6.47.1.10 INT db\_delete\_key1 (HANDLE *hDB*, HANDLE *hKey*, INT *level*, BOOL *follow\_links*)

Delete a subtree, using level information (only called internally by `db_delete_key()`(p.??))

For ~~Parameters~~ **Parameters** only.

*hDB* ODB handle obtained via `cm_get_experiment_database()`(p.??).

*hKey* Key handle to start with, 0 for root

*level* Recursion level, must be zero when

*follow\_links* Follow links when TRUE called from a user routine

**Returns:**

DB\_SUCCESS, DB\_INVALID\_HANDLE, DB\_OPEN\_RECORD

Definition at line 1944 of file odb.c.

Referenced by `cm_delete_client_info()`, and `db_delete_key()`.

#### 6.47.1.11 INT db\_enum\_key (HANDLE *hDB*, HANDLE *hKey*, INT *index*, HANDLE \* *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()`(p.??). 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;
HANDLE 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()`(p.??).

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

*index* Subkey index, should 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()**(p.??) and **db\_get\_data()**(p.??)

**Returns:**

DB\_SUCCESS, DB\_INVALID\_HANDLE, DB\_NO\_MORE\_SUBKEYS

Definition at line 3355 of file odb.c.

Referenced by **cm\_connect\_client()**, **cm\_exist()**, **cm\_set\_client\_info()**, **cm\_shutdown()**, **cm\_transition()**, **logger\_root()**, **main()**, and **update\_odb()**.

#### 6.47.1.12 INT db\_find\_key (HANDLE hDB, HANDLE hKey, char \* key\_name, HANDLE \* subhKey)

Returns key handle for a key with a specific name.

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

The function **db\_find\_key()**(p.??) must be used to convert a key name to a handle. Most other database functions use this key handle in various operations.

```
HANDLE 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()**(p.??).

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

*key\_name* Name of key to search, can contain directories.

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

**Returns:**

DB\_SUCCESS, DB\_INVALID\_HANDLE, DB\_NO\_ACCESS, DB\_NO\_KEY



Definition at line 2149 of file odb.c.

Referenced by al\_trigger\_alarm(), cm\_connect\_client(), cm\_exist(), cm\_get\_client\_info(), cm\_msg\_log(), cm\_msg\_log1(), cm\_msg\_retrieve(), cm\_register\_deferred\_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(), logger\_root(), main(), and update\_odb().

**6.47.1.13** INT db\_get\_data (HANDLE *hDB*, HANDLE *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.

```
HANDLE 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()(p. ??).

*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(p. ??)).

#### Returns:

DB\_SUCCESS, DB\_INVALID\_HANDLE, DB\_TRUNCATED, DB\_TYPE\_MISMATCH

Definition at line 4115 of file odb.c.

Referenced by cm\_connect\_client(), cm\_get\_client\_info(), cm\_set\_client\_info(), db\_copy(), and db\_get\_record().

**6.47.1.14** INT db\_get\_data\_index (HANDLE *hDB*, HANDLE *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(p.??).

*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(p.??)).

**Returns:**

DB\_SUCCESS, DB\_INVALID\_HANDLE, DB\_TRUNCATED, DB\_OUT\_OF\_RANGE

Definition at line 4346 of file odb.c.

**6.47.1.15** INT db\_get\_key (HANDLE *hDB*, HANDLE *hKey*, KEY \* *key*)

Get key structure from a handle.

KEY(p.??) 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;
HANDLE 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()**(p. ??).

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

**key** Pointer to **KEY**(p. ??) structure.

**Returns:**

DB\_SUCCESS, DB\_INVALID\_HANDLE

Definition at line 3687 of file odb.c.

Referenced by **cm\_check\_client()**, **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\_set\_record()**, **main()**, and **update\_odb()**.

**6.47.1.16** **INT db\_get\_key\_info (HANDLE hDB, HANDLE 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()**(p. ??).

**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**(p. ??)).

**num\_values** Number of values in key.

**item\_size** Size of individual key value (used for strings)

**Returns:**

DB\_SUCCESS, DB\_INVALID\_HANDLE

Definition at line 3805 of file odb.c.

#### 6.47.1.17 INT db\_get\_key\_time (HANDLE *hDB*, HANDLE *hKey*, DWORD \* *delta*)

Get time when key was last modified

##### Parameters:

*hDB* ODB handle obtained via `cm_get_experiment_database()(p.??)`.

*hKey* Handle of key to operate on

*delta* Seconds since last update

##### Returns:

DB\_SUCCESS, DB\_INVALID\_HANDLE

Definition at line 3747 of file odb.c.

#### 6.47.1.18 INT db\_get\_record (HANDLE *hDB*, HANDLE *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()(p.??)` 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()(p.??)` before `db_get_record()(p.??)` 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()
{
    HANDLE 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()**(p. ??).

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

**data** Pointer to the retrieved data.

**buf\_size** Size of data structure, must be obtained via **sizeof(RECORD-NAME)**.

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

**Returns:**

DB\_SUCCESS, DB\_INVALID\_HANDLE, DB\_STRUCT\_SIZE\_MISMATCH

Definition at line 6572 of file odb.c.

Referenced by **al\_trigger\_alarm()**, **cm\_transition()**, **db\_open\_record()**, **db\_update\_record()**, and **main()**.

#### 6.47.1.19 INT db\_get\_record\_size (HANDLE hDB, HANDLE hKey, INT align, INT \* buf\_size)

Calculates the size of a record.

**Parameters:**

**hDB** ODB handle obtained via **cm\_get\_experiment\_database()**(p. ??).

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

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

**buf\_size** Size of record structure

**Returns:**

DB\_SUCCESS, DB\_INVALID\_HANDLE, DB\_TYPE\_MISMATCH, DB\_STRUCT\_SIZE\_MISMATCH, DB\_NO\_KEY

Definition at line 6485 of file odb.c.

Referenced by **db\_get\_record()**, **db\_open\_record()**, and **db\_set\_record()**.

**6.47.1.20** INT db\_get\_value (HANDLE *hDB*, HANDLE *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()**(p. ??).

*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**(p. ??))

*create* If TRUE, create key if not existing

**Returns:**

DB\_SUCCESS, DB\_INVALID\_HANDLE, DB\_NO\_ACCESS, DB\_TYPE\_MISMATCH, DB\_TRUNCATED, DB\_NO\_KEY

Definition at line 3225 of file odb.c.

Referenced by **al\_trigger\_alarm()**, **cm\_check\_client()**, **cm\_connect\_experiment1()**, **cm\_exist()**, **cm\_msg\_log()**, **cm\_msg\_log1()**, **cm\_msg\_retrieve()**, **cm\_register\_deferred\_transition()**, **cm\_register\_transition()**, **cm\_set\_client\_info()**, **cm\_shutdown()**, **cm\_transition()**, **el\_submit()**, **logger\_root()**, **main()**, **scheduler()**, and **tr\_prestart()**.

#### 6.47.1.21 INT db\_load (HANDLE *hDB*, HANDLE *hKeyRoot*, char \* *filename*, BOOL *bRemote*)

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

This function is used by the ODBEdit command load. For a description of the ASCII format, see **db\_copy()**(p. ??). 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()**(p. ??).

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

**filename** Filename of .ODB file.

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

##### Returns:

DB\_SUCCESS, DB\_INVALID\_HANDLE, DB\_FILE\_ERROR

Definition at line 5115 of file odb.c.

#### 6.47.1.22 INT db\_lock\_database (HANDLE *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

Definition at line 1556 of file odb.c.

Referenced by **cm\_check\_client()**, **cm\_cleanup()**, **cm\_delete\_client\_info()**, **cm\_get\_watchdog\_info()**, **cm\_set\_watchdog\_params()**, **db\_close\_database()**, **db\_create\_key()**, **db\_create\_record()**, **db\_delete\_key1()**, **db\_enum\_key()**, **db\_find\_key()**, **db\_get\_data()**, **db\_get\_data\_index()**, **db\_get\_key()**, **db\_get\_key\_info()**, **db\_get\_key\_time()**, **db\_get\_record()**, **db\_get\_record\_size()**, **db\_get\_value()**, **db\_open\_database()**, **db\_set\_data()**, **db\_set\_data\_index()**, **db\_set\_record()**, and **db\_set\_value()**.

#### 6.47.1.23 INT db\_open\_database (char \* *database\_name*, INT *database\_size*, HANDLE \* *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()**(p. ??).

**Returns:**

DB\_SUCCESS, DB\_CREATED, DB\_INVALID\_NAME, DB\_NO\_MEMORY, DB\_MEMSIZE\_MISMATCH, DB\_NO\_MUTEX, DB\_INVALID\_PARAM, RPC\_NET\_ERROR

Definition at line 955 of file odb.c.

Referenced by **cm\_connect\_experiment1()**.

#### 6.47.1.24 INT db\_open\_record (HANDLE hDB, HANDLE hKey, void \* ptr, INT rec\_size, WORD access\_mode, void(\* dispatcher)(INT, INT, void \*), void \* info)

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

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

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

The access mode might either be **MODE\_READ** or **MODE\_WRITE**. In read mode, the ODB sub-tree is automatically copied to the local structure when modified by other clients. In write mode, the local structure is copied to the ODB sub-tree if it has been modified locally. This update has to be manually scheduled by calling **db\_send\_changed\_records()**(p. ??) 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()**(p. ??) 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()
{
    HANDLE 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()**(p.??).

**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: HANDLE hDB, HANDLE hKey, void \*info

**info** Additional info passed to the dispatcher.

#### Returns:

DB\_SUCCESS, DB\_INVALID\_HANDLE, DB\_NO\_MEMORY, DB\_NO\_ACCESS, DB\_STRUCT\_SIZE\_MISMATCH

Definition at line 7536 of file odb.c.

Referenced by cm\_register\_deferred\_transition().

#### 6.47.1.25 INT db\_paste (HANDLE *hDB*, HANDLE *hKeyRoot*, char \* *buffer*)

Copy an ODB subtree in ASCII format from a buffer

##### Parameters:

*hDB* ODB handle obtained via `cm_get_experiment_database()`(p. ??).

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

*buffer* NULL-terminated buffer

##### Returns:

DB\_SUCCESS, DB\_TRUNCATED, DB\_NO\_MEMORY

Definition at line 5452 of file odb.c.

Referenced by `db_create_record()`, and `db_load()`.

#### 6.47.1.26 INT db\_protect\_database (HANDLE *hDB*)

Protect a database for read/write access outside of the `db_XXX` functions

##### Parameters:

*hDB* ODB handle obtained via `cm_get_experiment_database()`(p. ??).

##### Returns:

DB\_SUCCESS, DB\_INVALID\_HANDLE

Definition at line 1620 of file odb.c.

#### 6.47.1.27 INT db\_save (HANDLE *hDB*, HANDLE *hKey*, char \* *filename*, BOOL *bRemote*)

Save a branch of a database to an .ODB file

This function is used by the ODBEdit command save. For a description of the ASCII format, see `db_copy()`(p. ??). 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()`(p. ??).

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

*filename* Filename of .ODB file.

*bRemote* Flag for saving database on remote server.

**Returns:**

DB\_SUCCESS, DB\_FILE\_ERROR

Definition at line 5863 of file odb.c.

**6.47.1.28 INT db\_save\_struct (HANDLE hDB, HANDLE hKey, char \* file\_name, char \* struct\_name, BOOL append)**

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

**Parameters:**

*hDB* ODB handle obtained via **cm\_get\_experiment\_database()**(p.??).

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

*file\_name* Filename of .ODB file.

*struct\_name* Name of structure. If struct\_name == NULL, the name of the key is used.

*append* If TRUE, append to end of existing file

**Returns:**

DB\_SUCCESS, DB\_INVALID\_HANDLE, DB\_FILE\_ERROR

Definition at line 5928 of file odb.c.

**6.47.1.29 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()**(p.??) 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 }

HANDLE 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 structure 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 7955 of file odb.c.

Referenced by scheduler().

#### 6.47.1.30 INT db\_set\_data (HANDLE *hDB*, HANDLE *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.

```

HANDLE 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()(p. ??).

*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**(p. ??)).

**Returns:**

DB\_SUCCESS, DB\_INVALID\_HANDLE, DB\_TRUNCATED

Definition at line 4475 of file odb.c.

Referenced by db\_paste(), db\_set\_record(), and update\_odb().

### 6.47.1.31 INT db\_set\_data\_index (HANDLE hDB, HANDLE 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()`(p.??).

*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**(p.??)).

#### Returns:

DB\_SUCCESS, DB\_INVALID\_HANDLE, DB\_NO\_ACCESS, DB\_TYPE\_MISMATCH

Definition at line 4702 of file odb.c.

### 6.47.1.32 INT db\_set\_record (HANDLE hDB, HANDLE 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()`(p.??) 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()`(p.??) 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()`**(p. ??).

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

***data*** Pointer where data is stored.

***buf\_size*** Size of data structure, must be obtained via `sizeof(RECORD-NAME)`.

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

**Returns:**

DB\_SUCCESS, DB\_INVALID\_HANDLE, DB\_TYPE\_MISMATCH, DB\_STRUCT\_SIZE\_MISMATCH

Definition at line 6677 of file odb.c.

Referenced by `al_trigger_alarm()`, `db_open_record()`, `db_send_changed_records()`, `main()`, `tr_prestart()`, and `update_odb()`.

**6.47.1.33** **INT** `db_set_value` (**HANDLE** *hDB*, **HANDLE** *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()`**(p. ??).

***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**(p.??))

**Returns:**

DB\_SUCCESS, DB\_INVALID\_HANDLE, DB\_NO\_ACCESS, DB\_TYPE\_MISMATCH

Definition at line 3090 of file odb.c.

Referenced by al\_trigger\_alarm(), cm\_connect\_experiment1(), cm\_delete\_client\_info(), cm\_register\_deferred\_transition(), cm\_register\_transition(), cm\_set\_client\_info(), cm\_set\_watchdog\_params(), cm\_transition(), db\_create\_link(), db\_get\_value(), main(), and update\_odb().

**6.47.1.34 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**(p.??)).

**Returns:**

DB\_SUCCESS

Definition at line 6103 of file odb.c.

Referenced by db\_copy().

**6.47.1.35 INT db\_unlock\_database (HANDLE *hDB*)**

Unlock a database via system mutex calls.

**Parameters:**

***hDB*** Handle to the database to unlock

**Returns:**

DB\_SUCCESS, DB\_INVALID\_HANDLE

Definition at line 1590 of file odb.c.

Referenced by cm\_check\_client(), cm\_cleanup(), cm\_delete\_client\_info(), cm\_get\_watchdog\_info(), cm\_set\_watchdog\_params(), db\_close\_database(), db\_create\_key(), db\_create\_record(), db\_delete\_key1(), db\_enum\_key(), db\_find\_key(), db\_get\_data(), db\_get\_data\_index(), db\_get\_key(), db\_get\_key\_info(), db\_get\_key\_time(), db\_get\_record(), db\_get\_record\_size(), db\_get\_value(), db\_open\_database(), db\_set\_data(), db\_set\_data\_index(), db\_set\_record(), and db\_set\_value().

**6.47.1.36 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 notification to the client.

**Parameters:**

***hDB*** ODB handle obtained via **cm\_get\_experiment\_database()**(p. ??).

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

***socket*** optional server socket

**Returns:**

DB\_SUCCESS, DB\_INVALID\_HANDLE

Definition at line 7744 of file odb.c.

**6.47.1.37 BOOL equal\_ustring (char \* *str1*, char \* *str2*)**

Definition at line 1650 of file 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\_find\_key(), db\_open\_database(), db\_paste(), logger\_root(), and main().

#### 6.47.1.38 char\* extract\_key (char \* *key\_list*, char \* *key\_name*)

Definition at line 1638 of file odb.c.

Referenced by db\_create\_key(), and db\_find\_key().



## Chapter 7

# Midas Data Structure Documentation

### 7.1 ALARM Struct Reference

#### 7.1.1 Detailed Description

Alarm structure

Definition at line 1583 of file midas.h.

#### 7.1.2 Field Documentation

##### 7.1.2.1 **BOOL ALARM::active**

Definition at line 1584 of file midas.h.

Referenced by `al_trigger_alarm()`.

##### 7.1.2.2 **char ALARM::alarm\_class[32]**

Definition at line 1592 of file midas.h.

Referenced by `al_trigger_alarm()`.

##### 7.1.2.3 **char ALARM::alarm\_message[80]**

Definition at line 1593 of file midas.h.

Referenced by `al_trigger_alarm()`.

#### **7.1.2.4 INT ALARM::check\_interval**

Definition at line 1587 of file `midas.h`.

Referenced by `al_trigger_alarm()`.

#### **7.1.2.5 DWORD ALARM::checked\_last**

Definition at line 1588 of file `midas.h`.

Referenced by `al_trigger_alarm()`.

#### **7.1.2.6 char ALARM::condition[256]**

Definition at line 1591 of file `midas.h`.

#### **7.1.2.7 char ALARM::time\_triggered\_first[32]**

Definition at line 1589 of file `midas.h`.

Referenced by `al_trigger_alarm()`.

#### **7.1.2.8 char ALARM::time\_triggered\_last[32]**

Definition at line 1590 of file `midas.h`.

Referenced by `al_trigger_alarm()`.

#### **7.1.2.9 INT ALARM::triggered**

Definition at line 1585 of file `midas.h`.

Referenced by `al_trigger_alarm()`.

#### **7.1.2.10 INT ALARM::type**

Definition at line 1586 of file `midas.h`.

Referenced by `al_trigger_alarm()`.

## 7.2 ALARM\_CLASS Struct Reference

### 7.2.1 Detailed Description

Alarm class structure

Definition at line 1553 of file midas.h.

### 7.2.2 Field Documentation

#### 7.2.2.1 char ALARM\_CLASS::display\_bgcolor[32]

Definition at line 1562 of file midas.h.

#### 7.2.2.2 char ALARM\_CLASS::display\_fgcolor[32]

Definition at line 1563 of file midas.h.

#### 7.2.2.3 char ALARM\_CLASS::execute\_command[256]

Definition at line 1558 of file midas.h.

#### 7.2.2.4 INT ALARM\_CLASS::execute\_interval

Definition at line 1559 of file midas.h.

#### 7.2.2.5 DWORD ALARM\_CLASS::execute\_last

Definition at line 1560 of file midas.h.

#### 7.2.2.6 BOOL ALARM\_CLASS::stop\_run

Definition at line 1561 of file midas.h.

#### 7.2.2.7 INT ALARM\_CLASS::system\_message\_interval

Definition at line 1556 of file midas.h.

#### 7.2.2.8 DWORD ALARM\_CLASS::system\_message\_last

Definition at line 1557 of file midas.h.

**7.2.2.9    BOOL ALARM\_CLASS::write\_elog\_message**

Definition at line 1555 of file midas.h.

**7.2.2.10    BOOL ALARM\_CLASS::write\_system\_message**

Definition at line 1554 of file midas.h.



## 7.3 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**

### 7.3.1 Field Documentation

#### 7.3.1.1 INT(\* ANA\_MODULE::analyzer)(EVENT\_HEADER \*, void \*)

Pointer to user analyzer routine

#### 7.3.1.2 char ANA\_MODULE::author[NAME\_LENGTH]

Author

Definition at line 1356 of file midas.h.

#### 7.3.1.3 INT(\* ANA\_MODULE::bor)(INT run\_number)

Pointer to begin-of-run routine

#### 7.3.1.4 BOOL ANA\_MODULE::enabled

Enabled flag

Definition at line 1365 of file midas.h.

#### 7.3.1.5 INT(\* ANA\_MODULE::eor)(INT run\_number)

Pointer to end-of-run routine

**7.3.1.6 INT(\* ANA\_MODULE::exit)()**

Pointer to exit routine

**7.3.1.7 INT(\* ANA\_MODULE::init)()**

Pointer to init routine

**7.3.1.8 char\*\* ANA\_MODULE::init\_str**

Parameter init string

Definition at line 1364 of file midas.h.

**7.3.1.9 char ANA\_MODULE::name[NAME\_LENGTH]**

Module name

Definition at line 1355 of file midas.h.

**7.3.1.10 INT ANA\_MODULE::param\_size**

Size of parameter structure

Definition at line 1363 of file midas.h.

**7.3.1.11 void\* ANA\_MODULE::parameters**

Pointer to parameter structure

Definition at line 1362 of file midas.h.

## 7.4 ANA\_TEST Struct Reference

### 7.4.1 Field Documentation

#### 7.4.1.1 DWORD ANA\_TEST::count

Definition at line 1415 of file midas.h.

#### 7.4.1.2 char ANA\_TEST::name[80]

Definition at line 1413 of file midas.h.

#### 7.4.1.3 BOOL ANA\_TEST::registered

Definition at line 1414 of file midas.h.

#### 7.4.1.4 BOOL ANA\_TEST::value

Definition at line 1416 of file midas.h.

## 7.5 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
- HANDLE buffer\_handle
- HANDLE request\_id
- HANDLE hkey\_variables
- HANDLE hkey\_common
- void \* addr
- DWORD events\_received
- DWORD events\_written

### 7.5.1 Field Documentation

#### 7.5.1.1 void\* ANALYZE\_REQUEST::addr

Buffer for CWNT filling

Definition at line 1398 of file midas.h.

#### 7.5.1.2 ANA\_MODULE\*\* ANALYZE\_REQUEST::ana\_module

List of analyzer modules

Definition at line 1388 of file midas.h.

#### 7.5.1.3 INT(\* ANALYZE\_REQUEST::analyzer)(EVENT\_HEADER \*, void \*)

Pointer to user analyzer routine

#### 7.5.1.4 AR\_INFO ANALYZE\_REQUEST::ar\_info

From above

Definition at line 1386 of file midas.h.

**7.5.1.5 AR\_STATS ANALYZE\_REQUEST::ar\_stats**

Definition at line 1406 of file midas.h.

**7.5.1.6 BANK\_LIST\* ANALYZE\_REQUEST::bank\_list**

List of banks for event

Definition at line 1389 of file midas.h.

**7.5.1.7 HANDLE ANALYZE\_REQUEST::buffer\_handle**

MIDAS buffer handle

Definition at line 1394 of file midas.h.

**7.5.1.8 char ANALYZE\_REQUEST::event\_name[NAME\_LENGTH]**

Event name

Definition at line 1385 of file midas.h.

**7.5.1.9 DWORD ANALYZE\_REQUEST::events\_received**

number of events sent

Definition at line 1404 of file midas.h.

**7.5.1.10 DWORD ANALYZE\_REQUEST::events\_written**

number of events written

Definition at line 1405 of file midas.h.

**7.5.1.11 HANDLE ANALYZE\_REQUEST::hkey\_common**

Key to common subtree

Definition at line 1397 of file midas.h.

**7.5.1.12 HANDLE ANALYZE\_REQUEST::hkey\_variables**

Key to variables subtree in ODB

Definition at line 1396 of file midas.h.

**7.5.1.13 char\*\* ANALYZE\_REQUEST::init\_string**

Definition at line 1392 of file midas.h.

**7.5.1.14 struct { ... } ANALYZE\_REQUEST::number**

**7.5.1.15 HANDLE ANALYZE\_REQUEST::request\_id**

Event request handle

Definition at line 1395 of file midas.h.

**7.5.1.16 DWORD ANALYZE\_REQUEST::run**

< Buffer for event number for CWNT

Definition at line 1400 of file midas.h.

**7.5.1.17 INT ANALYZE\_REQUEST::rwnt\_buffer\_size**

Size in events of RW N-tuple buf

Definition at line 1390 of file midas.h.

**7.5.1.18 DWORD ANALYZE\_REQUEST::serial**

Definition at line 1401 of file midas.h.

**7.5.1.19 INT ANALYZE\_REQUEST::status**

One of FE\_XXX

Definition at line 1393 of file midas.h.

**7.5.1.20 DWORD ANALYZE\_REQUEST::time**

Definition at line 1402 of file midas.h.

**7.5.1.21 BOOL ANALYZE\_REQUEST::use\_tests**

Use tests for this event

Definition at line 1391 of file midas.h.

## 7.6 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]

### 7.6.1 Field Documentation

#### 7.6.1.1 char AR\_INFO::buffer[NAME\_LENGTH]

Event buffer to send events into

Definition at line 1372 of file midas.h.

#### 7.6.1.2 char AR\_INFO::client\_name[NAME\_LENGTH]

Analyzer name

Definition at line 1374 of file midas.h.

#### 7.6.1.3 BOOL AR\_INFO::enabled

Enable flag

Definition at line 1373 of file midas.h.

#### 7.6.1.4 INT AR\_INFO::event\_id

Event ID associated with equipm.

Definition at line 1369 of file midas.h.

#### 7.6.1.5 char AR\_INFO::host[NAME\_LENGTH]

Host on which analyzer is running

Definition at line 1375 of file midas.h.



#### 7.6.1.6 INT AR\_INFO::sampling\_type

GET\_ALL/GET\_SOME

Definition at line 1371 of file midas.h.

#### 7.6.1.7 INT AR\_INFO::trigger\_mask

Trigger mask

Definition at line 1370 of file midas.h.

## 7.7 AR\_STATS Struct Reference

### 7.7.1 Field Documentation

#### 7.7.1.1 double AR\_STATS::events\_per\_sec

Definition at line 1380 of file midas.h.

#### 7.7.1.2 double AR\_STATS::events\_received

Definition at line 1379 of file midas.h.

#### 7.7.1.3 double AR\_STATS::events\_written

Definition at line 1381 of file midas.h.

## 7.8 BANK Struct Reference

### Data Fields

- char **name** [4]
- WORD **type**
- WORD **data\_size**

### 7.8.1 Field Documentation

#### 7.8.1.1 WORD BANK::data\_size

- 

Definition at line 1321 of file midas.h.

Referenced by `bk_close()`, `bk_create()`, `bk_find()`, `bk_iterate()`, `bk_locate()`, and `bk_swap()`.

#### 7.8.1.2 char BANK::name[4]

- 

Definition at line 1319 of file midas.h.

Referenced by `bk_close()`, `bk_create()`, `bk_find()`, `bk_list()`, `bk_locate()`, and `update_odb()`.

#### 7.8.1.3 WORD BANK::type

- 

Definition at line 1320 of file midas.h.

Referenced by `bk_close()`, `bk_create()`, `bk_find()`, `bk_locate()`, `bk_swap()`, and `update_odb()`.

## 7.9 BANK32 Struct Reference

### Data Fields

- char **name** [4]
- DWORD **type**
- DWORD **data\_size**

#### 7.9.1 Field Documentation

##### 7.9.1.1 DWORD BANK32::data\_size

- 

Definition at line 1327 of file midas.h.

Referenced by `bk_close()`, `bk_create()`, `bk_find()`, `bk_locate()`, and `bk_swap()`.

##### 7.9.1.2 char BANK32::name[4]

- 

Definition at line 1325 of file midas.h.

Referenced by `bk_close()`, `bk_create()`, `bk_find()`, `bk_list()`, `bk_locate()`, and `update_odb()`.

##### 7.9.1.3 DWORD BANK32::type

- 

Definition at line 1326 of file midas.h.

Referenced by `bk_close()`, `bk_create()`, `bk_find()`, `bk_locate()`, `bk_swap()`, and `update_odb()`.

## 7.10 BANK\_HEADER Struct Reference

### Data Fields

- **DWORD data\_size**
- **DWORD flags**

### 7.10.1 Field Documentation

#### 7.10.1.1 DWORD BANK\_HEADER::data\_size

Size in bytes

Definition at line 1314 of file midas.h.

Referenced by `bk_find()`, `bk_swap()`, and `eb_mfragment_add()`.

#### 7.10.1.2 DWORD BANK\_HEADER::flags

internal flag

Definition at line 1315 of file midas.h.

Referenced by `bk_swap()`.

## 7.11 BANK\_LIST Struct Reference

### Data Fields

- char **name** [9]
- WORD **type**
- DWORD **size**
- char \*\* **init\_str**
- BOOL **output\_flag**
- void \* **addr**
- DWORD **n\_data**
- HANDLE **def\_key**

### 7.11.1 Field Documentation

#### 7.11.1.1 void\* BANK\_LIST::addr

- 

Definition at line 1342 of file midas.h.

#### 7.11.1.2 HANDLE BANK\_LIST::def\_key

- 

Definition at line 1344 of file midas.h.

#### 7.11.1.3 char\*\* BANK\_LIST::init\_str

- 

Definition at line 1340 of file midas.h.

#### 7.11.1.4 DWORD BANK\_LIST::n\_data

- 

Definition at line 1343 of file midas.h.

**7.11.1.5 char BANK\_LIST::name[9]**

- 

Definition at line 1337 of file midas.h.

**7.11.1.6 BOOL BANK\_LIST::output\_flag**

- 

Definition at line 1341 of file midas.h.

**7.11.1.7 DWORD BANK\_LIST::size**

- 

Definition at line 1339 of file midas.h.

**7.11.1.8 WORD BANK\_LIST::type**

- 

Definition at line 1338 of file midas.h.

## 7.12 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`

### 7.12.1 Field Documentation

#### 7.12.1.1 **BOOL** `BUFFER::attached`

TRUE if buffer is attached

Definition at line 1184 of file `midas.h`.

Referenced by `bm_flush_cache()`, `bm_open_buffer()`, `bm_push_event()`, `bm_receive_event()`, `bm_send_event()`, and `bm_skip_event()`.

#### 7.12.1.2 **void**\* `BUFFER::buffer_data`

pointer to buffer data

Definition at line 1187 of file `midas.h`.

#### 7.12.1.3 **BUFFER\_HEADER**\* `BUFFER::buffer_header`

pointer to buffer header

Definition at line 1186 of file `midas.h`.

Referenced by `bm_empty_buffers()`, `bm_open_buffer()`, `bm_push_event()`, `bm_receive_event()`, and `bm_skip_event()`.



#### 7.12.1.4 **BOOL BUFFER::callback**

callback defined for this buffer

Definition at line 1199 of file midas.h.

Referenced by `bm_push_event()`.

#### 7.12.1.5 **INT BUFFER::client\_index**

index to CLIENT str. in buf.

Definition at line 1185 of file midas.h.

Referenced by `bm_close_buffer()`, `bm_empty_buffers()`, `bm_push_event()`, `bm_receive_event()`, `bm_skip_event()`, and `cm_set_watchdog_params()`.

#### 7.12.1.6 **INT BUFFER::index**

connection index / tid

Definition at line 1198 of file midas.h.

Referenced by `bm_open_buffer()`.

#### 7.12.1.7 **HNDLE BUFFER::mutex**

mutex/semaphore handle

Definition at line 1196 of file midas.h.

#### 7.12.1.8 **char\* BUFFER::read\_cache**

cache for burst read

Definition at line 1188 of file midas.h.

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

#### 7.12.1.9 **INT BUFFER::read\_cache\_rp**

cache read pointer

Definition at line 1190 of file midas.h.

Referenced by `bm_empty_buffers()`, `bm_push_event()`, `bm_receive_event()`, `bm_set_cache_size()`, and `bm_skip_event()`.

**7.12.1.10 INT BUFFER::read\_cache\_size**

cache size in bytes

Definition at line 1189 of file midas.h.

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

**7.12.1.11 INT BUFFER::read\_cache\_wp**

cache write pointer

Definition at line 1191 of file midas.h.

Referenced by `bm_empty_buffers()`, `bm_push_event()`, `bm_receive_event()`, `bm_set_cache_size()`, and `bm_skip_event()`.

**7.12.1.12 INT BUFFER::shm\_handle**

handle to shared memory

Definition at line 1197 of file midas.h.

**7.12.1.13 char\* BUFFER::write\_cache**

cache for burst read

Definition at line 1192 of file midas.h.

Referenced by `bm_flush_cache()`, `bm_send_event()`, and `bm_set_cache_size()`.

**7.12.1.14 INT BUFFER::write\_cache\_rp**

cache read pointer

Definition at line 1194 of file midas.h.

Referenced by `bm_flush_cache()`, and `bm_set_cache_size()`.

**7.12.1.15 INT BUFFER::write\_cache\_size**

cache size in bytes

Definition at line 1193 of file midas.h.

Referenced by `bm_flush_cache()`, `bm_send_event()`, and `bm_set_cache_size()`.

#### 7.12.1.16 INT BUFFER::write\_cache\_wp

cache write pointer

Definition at line 1195 of file `midas.h`.

Referenced by `bm_flush_cache()`, `bm_send_event()`, and `bm_set_cache_size()`.

## 7.13 BUFFER\_CLIENT Struct Reference

### Data Fields

- char **name** [NAME\_LENGTH]
- INT **pid**
- INT **tid**
- INT **thandle**
- INT **port**
- INT **read\_pointer**
- INT **max\_request\_index**
- INT **num\_received\_events**
- INT **num\_sent\_events**
- INT **num\_waiting\_events**
- float **data\_rate**
- BOOL **read\_wait**
- INT **write\_wait**
- BOOL **wake\_up**
- BOOL **all\_flag**
- DWORD **last\_activity**
- DWORD **watchdog\_timeout**

### 7.13.1 Field Documentation

#### 7.13.1.1 BOOL BUFFER\_CLIENT::all\_flag

at least one GET\_ALL request

Definition at line 1159 of file midas.h.

Referenced by `bm_remove_event_request()`.

#### 7.13.1.2 float BUFFER\_CLIENT::data\_rate

data rate in kB/sec

Definition at line 1155 of file midas.h.

#### 7.13.1.3 EVENT\_REQUEST BUFFER\_CLIENT::event\_request[MAX\_EVENT\_REQUESTS]

Definition at line 1163 of file midas.h.

Referenced by `bm_push_event()`, `bm_receive_event()`, `bm_remove_event_request()`, and `bm_send_event()`.

**7.13.1.4   DWORD BUFFER\_CLIENT::last\_activity**

time of last activity

Definition at line 1160 of file midas.h.

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

**7.13.1.5   INT BUFFER\_CLIENT::max\_request\_index**

index of last request

Definition at line 1151 of file midas.h.

Referenced by `bm_push_event()`, `bm_receive_event()`, and `bm_remove_event_request()`.

**7.13.1.6   char BUFFER\_CLIENT::name[NAME\_LENGTH]**

name of client

Definition at line 1145 of file midas.h.

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

**7.13.1.7   INT BUFFER\_CLIENT::num\_received\_events**

no of received events

Definition at line 1152 of file midas.h.

**7.13.1.8   INT BUFFER\_CLIENT::num\_sent\_events**

no of sent events

Definition at line 1153 of file midas.h.

**7.13.1.9   INT BUFFER\_CLIENT::num\_waiting\_events**

no of waiting events

Definition at line 1154 of file midas.h.

**7.13.1.10   INT BUFFER\_CLIENT::pid**

process ID

Definition at line 1146 of file midas.h.

Referenced by `bm_close_buffer()`, `bm_flush_cache()`, `bm_open_buffer()`, `bm_send_event()`, and `cm_cleanup()`.

#### **7.13.1.11 INT BUFFER\_CLIENT::port**

UDP port for wake up

Definition at line 1149 of file midas.h.

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

#### **7.13.1.12 INT BUFFER\_CLIENT::read\_pointer**

read pointer to buffer

Definition at line 1150 of file midas.h.

Referenced by `bm_empty_buffers()`, `bm_open_buffer()`, `bm_push_event()`, `bm_receive_event()`, and `bm_skip_event()`.

#### **7.13.1.13 BOOL BUFFER\_CLIENT::read\_wait**

wait for read - flag

Definition at line 1156 of file midas.h.

Referenced by `bm_close_buffer()`, `bm_flush_cache()`, `bm_receive_event()`, and `cm_cleanup()`.

#### **7.13.1.14 INT BUFFER\_CLIENT::thandle**

thread handle

Definition at line 1148 of file midas.h.

Referenced by `bm_open_buffer()`.

#### **7.13.1.15 INT BUFFER\_CLIENT::tid**

thread ID

Definition at line 1147 of file midas.h.

Referenced by `bm_open_buffer()`.

**7.13.1.16    BOOL BUFFER\_CLIENT::wake\_up**

client got a wake-up msg

Definition at line 1158 of file midas.h.

**7.13.1.17    DWORD BUFFER\_CLIENT::watchdog\_timeout**

timeout in ms

Definition at line 1161 of file midas.h.

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

**7.13.1.18    INT BUFFER\_CLIENT::write\_wait**

wait for write # bytes

Definition at line 1157 of file midas.h.

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

## 7.14 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]

### 7.14.1 Field Documentation

#### 7.14.1.1 **BUFFER\_CLIENT** **BUFFER\_HEADER::client**[MAX\_CLIENTS]

entries for clients

Definition at line 1177 of file midas.h.

Referenced by `bm_close_buffer()`, `bm_flush_cache()`, `bm_open_buffer()`, `bm_push_event()`, `bm_receive_event()`, `bm_send_event()`, `bm_skip_event()`, `cm_cleanup()`, and `cm_set_watchdog_params()`.

#### 7.14.1.2 **INT** **BUFFER\_HEADER::max\_client\_index**

index of last client

Definition at line 1170 of file midas.h.

Referenced by `bm_close_buffer()`, `bm_flush_cache()`, `bm_open_buffer()`, `bm_push_event()`, `bm_receive_event()`, `bm_send_event()`, and `cm_cleanup()`.

#### 7.14.1.3 **char** **BUFFER\_HEADER::name**[NAME\_LENGTH]

name of buffer

Definition at line 1168 of file midas.h.

Referenced by `bm_check_buffers()`, `bm_close_buffer()`, `bm_flush_cache()`, `bm_open_buffer()`, `bm_push_event()`, `bm_send_event()`, and `cm_cleanup()`.



**7.14.1.4 INT BUFFER\_HEADER::num\_clients**

no of active clients

Definition at line 1169 of file midas.h.

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

**7.14.1.5 INT BUFFER\_HEADER::num\_in\_events**

no of received events

Definition at line 1174 of file midas.h.

Referenced by `bm_flush_cache()`, and `bm_send_event()`.

**7.14.1.6 INT BUFFER\_HEADER::num\_out\_events**

no of distributed events

Definition at line 1175 of file midas.h.

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

**7.14.1.7 INT BUFFER\_HEADER::read\_pointer**

read pointer

Definition at line 1172 of file midas.h.

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

**7.14.1.8 INT BUFFER\_HEADER::size**

size of data area in bytes

Definition at line 1171 of file midas.h.

Referenced by `bm_flush_cache()`, `bm_open_buffer()`, `bm_push_event()`, `bm_receive_event()`, and `bm_send_event()`.

**7.14.1.9 INT BUFFER\_HEADER::write\_pointer**

read pointer

Definition at line 1173 of file midas.h.

Referenced by `bm_flush_cache()`, `bm_open_buffer()`, `bm_push_event()`, `bm_receive_event()`, `bm_send_event()`, and `bm_skip_event()`.

## 7.15 BUS\_DRIVER Struct Reference

### Data Fields

- char **name** [NAME\_LENGTH]
- INT(\* **bd** )(INT cmd,...)
- void \* **bd\_info**

### 7.15.1 Field Documentation

#### 7.15.1.1 INT(\* BUS\_DRIVER::bd)(INT cmd, ...)

Device driver entry point

#### 7.15.1.2 void\* BUS\_DRIVER::bd\_info

Private info for bus driver

Definition at line 1240 of file midas.h.

#### 7.15.1.3 char BUS\_DRIVER::name[NAME\_LENGTH]

Driver name

Definition at line 1238 of file midas.h.

## 7.16 DATABASE Struct Reference

### 7.16.1 Field Documentation

#### 7.16.1.1 **BOOL DATABASE::attached**

Definition at line 480 of file msystem.h.

#### 7.16.1.2 **INT DATABASE::client\_index**

Definition at line 481 of file msystem.h.

Referenced by `cm_set_watchdog_params()`.

#### 7.16.1.3 **void\* DATABASE::database\_data**

Definition at line 483 of file msystem.h.

#### 7.16.1.4 **DATABASE\_HEADER\* DATABASE::database\_header**

Definition at line 482 of file msystem.h.

#### 7.16.1.5 **INT DATABASE::index**

Definition at line 487 of file msystem.h.

#### 7.16.1.6 **INT DATABASE::lock\_cnt**

Definition at line 485 of file msystem.h.

#### 7.16.1.7 **HNDLE DATABASE::mutex**

Definition at line 484 of file msystem.h.

#### 7.16.1.8 **char DATABASE::name[NAME\_LENGTH]**

Definition at line 479 of file msystem.h.

#### 7.16.1.9 **BOOL DATABASE::protect**

Definition at line 488 of file msystem.h.

**7.16.1.10   HANDLE DATABASE::shm\_handle**

Definition at line 486 of file msystem.h.

## 7.17 DATABASE\_CLIENT Struct Reference

### 7.17.1 Field Documentation

#### 7.17.1.1 DWORD DATABASE\_CLIENT::last\_activity

Definition at line 454 of file msystem.h.

Referenced by cm\_cleanup(), cm\_get\_watchdog\_info(), cm\_set\_watchdog\_params(), and db\_open\_database().

#### 7.17.1.2 INT DATABASE\_CLIENT::max\_index

Definition at line 456 of file msystem.h.

Referenced by db\_close\_database().

#### 7.17.1.3 char DATABASE\_CLIENT::name[NAME\_LENGTH]

Definition at line 448 of file msystem.h.

Referenced by cm\_cleanup(), cm\_get\_watchdog\_info(), and db\_open\_database().

#### 7.17.1.4 INT DATABASE\_CLIENT::num\_open\_records

Definition at line 453 of file msystem.h.

Referenced by db\_open\_database().

#### 7.17.1.5 OPEN\_RECORD DATABASE\_CLIENT::open\_record[MAX\_OPEN\_RECORDS]

Definition at line 458 of file msystem.h.

Referenced by cm\_cleanup(), and db\_close\_database().

#### 7.17.1.6 INT DATABASE\_CLIENT::pid

Definition at line 449 of file msystem.h.

Referenced by cm\_cleanup(), cm\_get\_watchdog\_info(), db\_close\_database(), and db\_open\_database().

**7.17.1.7 INT DATABASE\_CLIENT::port**

Definition at line 452 of file msystem.h.

Referenced by db\_open\_database().

**7.17.1.8 INT DATABASE\_CLIENT::thandle**

Definition at line 451 of file msystem.h.

Referenced by db\_open\_database().

**7.17.1.9 INT DATABASE\_CLIENT::tid**

Definition at line 450 of file msystem.h.

Referenced by cm\_check\_client(), cm\_cleanup(), and db\_open\_database().

**7.17.1.10 DWORD DATABASE\_CLIENT::watchdog\_timeout**

Definition at line 455 of file msystem.h.

Referenced by cm\_cleanup(), cm\_get\_watchdog\_info(), cm\_set\_watchdog\_params(), and db\_open\_database().

## 7.18 DATABASE\_HEADER Struct Reference

### 7.18.1 Field Documentation

#### 7.18.1.1 DATABASE\_CLIENT DATABASE\_HEADER::client[MAX\_CLIENTS]

Definition at line 472 of file msystem.h.

Referenced by cm\_check\_client(), cm\_cleanup(), cm\_get\_watchdog\_info(), cm\_set\_watchdog\_params(), db\_close\_database(), and db\_open\_database().

#### 7.18.1.2 INT DATABASE\_HEADER::data\_size

Definition at line 467 of file msystem.h.

Referenced by db\_close\_database(), and db\_open\_database().

#### 7.18.1.3 INT DATABASE\_HEADER::first\_free\_data

Definition at line 470 of file msystem.h.

Referenced by db\_open\_database().

#### 7.18.1.4 INT DATABASE\_HEADER::first\_free\_key

Definition at line 469 of file msystem.h.

Referenced by db\_open\_database().

#### 7.18.1.5 INT DATABASE\_HEADER::key\_size

Definition at line 466 of file msystem.h.

Referenced by db\_open\_database().

#### 7.18.1.6 INT DATABASE\_HEADER::max\_client\_index

Definition at line 465 of file msystem.h.

Referenced by cm\_check\_client(), cm\_cleanup(), cm\_get\_watchdog\_info(), db\_close\_database(), and db\_open\_database().



**7.18.1.7 char DATABASE\_HEADER::name[NAME\_LENGTH]**

Definition at line 463 of file msystem.h.

Referenced by cm\_cleanup(), db\_close\_database(), and db\_open\_database().

**7.18.1.8 INT DATABASE\_HEADER::num\_clients**

Definition at line 464 of file msystem.h.

Referenced by cm\_cleanup(), db\_close\_database(), and db\_open\_database().

**7.18.1.9 INT DATABASE\_HEADER::root\_key**

Definition at line 468 of file msystem.h.

Referenced by db\_create\_key(), db\_delete\_key1(), db\_enum\_key(), db\_find\_key(), and db\_open\_database().

## 7.19 DEF\_RECORD Struct Reference

### 7.19.1 Field Documentation

#### 7.19.1.1 DWORD DEF\_RECORD::def\_offset

Definition at line 1450 of file midas.h.

#### 7.19.1.2 DWORD DEF\_RECORD::event\_id

Definition at line 1448 of file midas.h.

#### 7.19.1.3 char DEF\_RECORD::event\_name[NAME\_LENGTH]

Definition at line 1449 of file midas.h.

## 7.20 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**

### 7.20.1 Field Documentation

#### 7.20.1.1 INT(\* DEVICE\_DRIVER::bd)(INT cmd, ...)

Bus driver entry point

#### 7.20.1.2 INT DEVICE\_DRIVER::channels

Number of channels

Definition at line 1246 of file midas.h.

#### 7.20.1.3 INT(\* DEVICE\_DRIVER::dd)(INT cmd, ...)

Device driver entry point

#### 7.20.1.4 void\* DEVICE\_DRIVER::dd\_info

Private info for device driver

Definition at line 1249 of file midas.h.

#### 7.20.1.5 DWORD DEVICE\_DRIVER::flags

Combination of DF\_xx

Definition at line 1248 of file midas.h.

#### 7.20.1.6 char DEVICE\_DRIVER::name[NAME\_LENGTH]

Driver name

Definition at line 1244 of file midas.h.

## 7.21 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**
- HANDLE **buffer\_handle**
- HANDLE **hkey\_variables**
- DWORD **serial\_number**
- DWORD **subevent\_number**
- DWORD **odb\_out**
- DWORD **odb\_in**
- DWORD **bytes\_sent**
- DWORD **events\_sent**

### 7.21.1 Field Documentation

#### 7.21.1.1 HANDLE eqpmnt::buffer\_handle

MIDAS buffer handle

Definition at line 1291 of file midas.h.

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

#### 7.21.1.2 DWORD eqpmnt::bytes\_sent

number of bytes sent

Definition at line 1297 of file midas.h.

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

**7.21.1.3 INT(\* eqpmnt::cd)(INT cmd, PEQUIPMENT)**

Class driver routine

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

**7.21.1.4 void\* eqpmnt::cd\_info**

private data for class driver

Definition at line 1285 of file `midas.h`.

**7.21.1.5 DEVICE\_DRIVER\* eqpmnt::driver**

Device driver list

Definition at line 1283 of file `midas.h`.

**7.21.1.6 void\* eqpmnt::event\_descrip**

Init string for fixed events or bank list

Definition at line 1284 of file `midas.h`.

**7.21.1.7 DWORD eqpmnt::events\_sent**

number of events sent

Definition at line 1298 of file `midas.h`.

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

**7.21.1.8 INT eqpmnt::format**

FORMAT\_xxx

Definition at line 1290 of file `midas.h`.

Referenced by `scheduler()`.

**7.21.1.9 HANDLE eqpmnt::hkey\_variables**

Key to variables subtree in ODB

Definition at line 1292 of file `midas.h`.

Referenced by `scheduler()`.

**7.21.1.10 EQUIPMENT\_INFO eqpmnt::info**

From above

Definition at line 1280 of file midas.h.

Referenced by display(), interrupt\_routine(), main(), scheduler(), send\_all\_periodic\_events(), and send\_event().

**7.21.1.11 DWORD eqpmnt::last\_called**

Last time event was read

Definition at line 1287 of file midas.h.

Referenced by interrupt\_routine(), scheduler(), and send\_event().

**7.21.1.12 DWORD eqpmnt::last\_idle**

Last time idle func was called

Definition at line 1288 of file midas.h.

Referenced by scheduler().

**7.21.1.13 char eqpmnt::name[NAME\_LENGTH]**

Equipment name

Definition at line 1279 of file midas.h.

Referenced by display(), main(), scheduler(), send\_all\_periodic\_events(), and send\_event().

**7.21.1.14 DWORD eqpmnt::odb\_in**

# updated ODB -> FE

Definition at line 1296 of file midas.h.

**7.21.1.15 DWORD eqpmnt::odb\_out**

# updates FE -> ODB

Definition at line 1295 of file midas.h.

Referenced by interrupt\_routine(), scheduler(), and send\_event().

**7.21.1.16    DWORD eqpmnt::poll\_count**

Needed to poll 'period'

Definition at line 1289 of file midas.h.

Referenced by scheduler().

**7.21.1.17    INT(\* eqpmnt::readout)(char \*, INT)**

Pointer to user readout routine

Referenced by interrupt\_routine(), scheduler(), and send\_event().

**7.21.1.18    DWORD eqpmnt::serial\_number**

event serial number

Definition at line 1293 of file midas.h.

Referenced by interrupt\_routine(), scheduler(), and send\_event().

**7.21.1.19    EQUIPMENT\_STATS eqpmnt::stats**

Definition at line 1299 of file midas.h.

Referenced by display(), scheduler(), and send\_event().

**7.21.1.20    INT eqpmnt::status**

One of FE\_XXX

Definition at line 1286 of file midas.h.

Referenced by display(), main(), scheduler(), and send\_all\_periodic\_events().

**7.21.1.21    DWORD eqpmnt::subevent\_number**

subevent number

Definition at line 1294 of file midas.h.

Referenced by scheduler().

## 7.22 EQUIPMENT\_INFO Struct Reference

### Data Fields

- WORD **event\_id**
- WORD **trigger\_mask**
- char **buffer** [NAME\_LENGTH]
- INT **eq\_type**
- INT **source**
- char **format** [8]
- BOOL **enabled**
- INT **read\_on**
- INT **period**
- double **event\_limit**
- DWORD **num\_subevents**
- INT **history**
- char **frontend\_host** [NAME\_LENGTH]
- char **frontend\_name** [NAME\_LENGTH]
- char **frontend\_file\_name** [256]

### 7.22.1 Field Documentation

#### 7.22.1.1 char EQUIPMENT\_INFO::buffer[NAME\_LENGTH]

Event buffer to send events into

Definition at line 1255 of file midas.h.

#### 7.22.1.2 BOOL EQUIPMENT\_INFO::enabled

Enable flag

Definition at line 1259 of file midas.h.

Referenced by `display()`, `scheduler()`, and `send_all_periodic_events()`.

#### 7.22.1.3 INT EQUIPMENT\_INFO::eq\_type

One of EQ\_XXX

Definition at line 1256 of file midas.h.

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



**7.22.1.4 WORD EQUIPMENT\_INFO::event\_id**

Event ID associated with equipm.

Definition at line 1253 of file midas.h.

Referenced by interrupt\_routine(), scheduler(), and send\_event().

**7.22.1.5 double EQUIPMENT\_INFO::event\_limit**

Stop run when limit is reached

Definition at line 1262 of file midas.h.

Referenced by scheduler().

**7.22.1.6 char EQUIPMENT\_INFO::format[8]**

Data format to produce

Definition at line 1258 of file midas.h.

**7.22.1.7 char EQUIPMENT\_INFO::frontend\_file\_name[256]**

Source file used for user FE

Definition at line 1267 of file midas.h.

**7.22.1.8 char EQUIPMENT\_INFO::frontend\_host[NAME\_LENGTH]**

Host on which FE is running

Definition at line 1265 of file midas.h.

**7.22.1.9 char EQUIPMENT\_INFO::frontend\_name[NAME\_LENGTH]**

Frontend name

Definition at line 1266 of file midas.h.

**7.22.1.10 INT EQUIPMENT\_INFO::history**

Log history

Definition at line 1264 of file midas.h.

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

#### **7.22.1.11   DWORD EQUIPMENT\_INFO::num\_subevents**

Number of events in super event

Definition at line 1263 of file `midas.h`.

Referenced by `scheduler()`.

#### **7.22.1.12   INT EQUIPMENT\_INFO::period**

Readout interval/Polling time in ms

Definition at line 1261 of file `midas.h`.

Referenced by `scheduler()`.

#### **7.22.1.13   INT EQUIPMENT\_INFO::read\_on**

Combination of Read-On flags `RO_XXX`

Definition at line 1260 of file `midas.h`.

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

#### **7.22.1.14   INT EQUIPMENT\_INFO::source**

Event source (LAM/IRQ)

Definition at line 1257 of file `midas.h`.

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

#### **7.22.1.15   WORD EQUIPMENT\_INFO::trigger\_mask**

Trigger mask

Definition at line 1254 of file `midas.h`.

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

## 7.23 EQUIPMENT\_STATS Struct Reference

### 7.23.1 Field Documentation

#### 7.23.1.1 double EQUIPMENT\_STATS::events\_per\_sec

Definition at line 1272 of file midas.h.

Referenced by scheduler().

#### 7.23.1.2 double EQUIPMENT\_STATS::events\_sent

Definition at line 1271 of file midas.h.

Referenced by display(), scheduler(), and send\_event().

#### 7.23.1.3 double EQUIPMENT\_STATS::kbytes\_per\_sec

Definition at line 1273 of file midas.h.

Referenced by scheduler().

## 7.24 EVENT\_HEADER Struct Reference

### 7.24.1 Detailed Description

Event header

Definition at line 1079 of file midas.h.

#### Data Fields

- short int **event\_id**
- short int **trigger\_mask**
- DWORD **serial\_number**
- DWORD **time\_stamp**
- DWORD **data\_size**

### 7.24.2 Field Documentation

#### 7.24.2.1 DWORD EVENT\_HEADER::data\_size

size of event in bytes w/o header

Definition at line 1084 of file midas.h.

Referenced by `bm_compose_event()`, `bm_flush_cache()`, `bm_push_event()`, `bm_receive_event()`, `cm_msg()`, `cm_msg1()`, `interrupt_routine()`, `scheduler()`, `send_event()`, and `update_odb()`.

#### 7.24.2.2 short int EVENT\_HEADER::event\_id

event ID starting from one

Definition at line 1080 of file midas.h.

Referenced by `bm_compose_event()`, `bm_match_event()`, `bm_push_event()`, `bm_receive_event()`, `interrupt_routine()`, `scheduler()`, `send_event()`, and `update_odb()`.

#### 7.24.2.3 DWORD EVENT\_HEADER::serial\_number

serial number starting from one

Definition at line 1082 of file midas.h.

Referenced by `bm_compose_event()`, `bm_receive_event()`, `eb_user()`, `interrupt_routine()`, `scheduler()`, and `send_event()`.

#### 7.24.2.4 DWORD EVENT\_HEADER::time\_stamp

time of production of event

Definition at line 1083 of file midas.h.

Referenced by bm\_compose\_event(), bm\_receive\_event(), interrupt\_routine(), scheduler(), and send\_event().

#### 7.24.2.5 short int EVENT\_HEADER::trigger\_mask

hardware trigger mask

Definition at line 1081 of file midas.h.

Referenced by bm\_compose\_event(), bm\_match\_event(), bm\_receive\_event(), eb\_user(), interrupt\_routine(), scheduler(), and send\_event().

## 7.25 EVENT\_REQUEST Struct Reference

### 7.25.1 Detailed Description

Buffer structure

Definition at line 1134 of file midas.h.

#### Data Fields

- INT **id**
- BOOL **valid**
- short int **event\_id**
- short int **trigger\_mask**
- INT **sampling\_type**

### 7.25.2 Field Documentation

**7.25.2.1** `void(* EVENT_REQUEST::dispatch)(HANDLE, HANDLE, EVENT_HEADER *, void *)`

**7.25.2.2** `short int EVENT_REQUEST::event_id`

event ID

Definition at line 1137 of file midas.h.

Referenced by `bm_push_event()`, `bm_receive_event()`, and `bm_send_event()`.

**7.25.2.3** `INT EVENT_REQUEST::id`

request id

Definition at line 1135 of file midas.h.

Referenced by `bm_flush_cache()`, `bm_remove_event_request()`, and `bm_send_event()`.

**7.25.2.4** `INT EVENT_REQUEST::sampling_type`

dispatch function

Definition at line 1139 of file midas.h.

Referenced by `bm_flush_cache()`, `bm_remove_event_request()`, and `bm_send_event()`.

#### 7.25.2.5 short int EVENT\_REQUEST::trigger\_mask

trigger mask

Definition at line 1138 of file `midas.h`.

Referenced by `bm_push_event()`, `bm_receive_event()`, and `bm_send_event()`.

#### 7.25.2.6 BOOL EVENT\_REQUEST::valid

indicating a valid entry

Definition at line 1136 of file `midas.h`.

Referenced by `bm_flush_cache()`, `bm_push_event()`, `bm_receive_event()`, `bm_remove_event_request()`, and `bm_send_event()`.

## 7.26 FREE\_DESCRIP Struct Reference

### Data Fields

- INT `size`
- INT `next_free`

### 7.26.1 Field Documentation

#### 7.26.1.1 INT `FREE_DESCRIP::next_free`

Address of next free block

Definition at line 437 of file `mssystem.h`.

Referenced by `db_open_database()`.

#### 7.26.1.2 INT `FREE_DESCRIP::size`

size in bytes

Definition at line 436 of file `mssystem.h`.

Referenced by `db_open_database()`.



## 7.27 HIST\_RECORD Struct Reference

### 7.27.1 Field Documentation

#### 7.27.1.1 DWORD HIST\_RECORD::data\_size

Definition at line 1444 of file midas.h.

#### 7.27.1.2 DWORD HIST\_RECORD::def\_offset

Definition at line 1443 of file midas.h.

#### 7.27.1.3 DWORD HIST\_RECORD::event\_id

Definition at line 1441 of file midas.h.

#### 7.27.1.4 DWORD HIST\_RECORD::record\_type

Definition at line 1440 of file midas.h.

#### 7.27.1.5 DWORD HIST\_RECORD::time

Definition at line 1442 of file midas.h.

## 7.28 HISTORY Struct Reference

### 7.28.1 Field Documentation

#### 7.28.1.1 DWORD HISTORY::base\_time

Definition at line 1467 of file midas.h.

#### 7.28.1.2 DWORD HISTORY::def\_fh

Definition at line 1466 of file midas.h.

#### 7.28.1.3 DWORD HISTORY::def\_offset

Definition at line 1468 of file midas.h.

#### 7.28.1.4 DWORD HISTORY::event\_id

Definition at line 1460 of file midas.h.

#### 7.28.1.5 char HISTORY::event\_name[NAME\_LENGTH]

Definition at line 1461 of file midas.h.

#### 7.28.1.6 DWORD HISTORY::hist\_fh

Definition at line 1464 of file midas.h.

#### 7.28.1.7 DWORD HISTORY::index\_fh

Definition at line 1465 of file midas.h.

#### 7.28.1.8 DWORD HISTORY::n\_tag

Definition at line 1462 of file midas.h.

#### 7.28.1.9 TAG\* HISTORY::tag

Definition at line 1463 of file midas.h.

## 7.29 INDEX\_RECORD Struct Reference

### 7.29.1 Field Documentation

#### 7.29.1.1 DWORD INDEX\_RECORD::event\_id

Definition at line 1454 of file midas.h.

#### 7.29.1.2 DWORD INDEX\_RECORD::offset

Definition at line 1456 of file midas.h.

#### 7.29.1.3 DWORD INDEX\_RECORD::time

Definition at line 1455 of file midas.h.

## 7.30 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**

### 7.30.1 Field Documentation

#### 7.30.1.1 WORD KEY::access\_mode

Access mode

Definition at line 1210 of file midas.h.

Referenced by `db_create_key()`, `db_delete_key1()`, `db_find_key()`, `db_get_data()`, `db_get_data_index()`, `db_get_value()`, `db_open_database()`, `db_open_record()`, `db_set_data()`, `db_set_data_index()`, and `db_set_value()`.

#### 7.30.1.2 INT KEY::data

Address of variable (offset)

Definition at line 1207 of file midas.h.

Referenced by `db_create_key()`, `db_delete_key1()`, `db_enum_key()`, `db_find_key()`, `db_get_data()`, `db_get_data_index()`, `db_get_key_info()`, `db_get_value()`, `db_open_database()`, `db_set_data()`, `db_set_data_index()`, and `db_set_value()`.

#### 7.30.1.3 INT KEY::item\_size

Size of single data item

Definition at line 1209 of file midas.h.

Referenced by `db_copy()`, `db_create_key()`, `db_get_data()`, `db_get_data_index()`, `db_get_key_info()`, `db_get_record()`, `db_get_record_size()`, `db_get_value()`, `db_open_database()`, `db_set_data()`, `db_set_data_index()`, `db_set_record()`, `db_set_value()`, and `update_odb()`.

#### 7.30.1.4 INT KEY::last\_written

Time of last write action

Definition at line 1214 of file `midas.h`.

Referenced by `db_get_key_time()`, `db_set_data()`, `db_set_data_index()`, and `db_set_value()`.

#### 7.30.1.5 char KEY::name[NAME\_LENGTH]

name of variable

Definition at line 1206 of file `midas.h`.

Referenced by `cm_check_client()`, `cm_shutdown()`, `cm_transition()`, `db_check_record()`, `db_copy()`, `db_create_key()`, `db_find_key()`, `db_get_data()`, `db_get_data_index()`, `db_get_key_info()`, `db_get_record()`, `db_open_database()`, `db_save_struct()`, `db_set_data()`, `db_set_data_index()`, `db_set_record()`, and `main()`.

#### 7.30.1.6 INT KEY::next\_key

Address of next key

Definition at line 1212 of file `midas.h`.

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

#### 7.30.1.7 WORD KEY::notify\_count

Notify counter

Definition at line 1211 of file `midas.h`.

Referenced by `cm_cleanup()`, and `db_delete_key1()`.

#### 7.30.1.8 INT KEY::num\_values

number of values

Definition at line 1205 of file midas.h.

Referenced by 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\_set\_data(), db\_set\_data\_index(), db\_set\_record(), db\_set\_value(), and update\_odb().

#### **7.30.1.9 INT KEY::parent\_keylist**

keylist to which this key belongs

Definition at line 1213 of file midas.h.

Referenced by db\_create\_key(), db\_delete\_key1(), db\_enum\_key(), db\_find\_key(), and db\_open\_database().

#### **7.30.1.10 INT KEY::total\_size**

Total size of data block

Definition at line 1208 of file midas.h.

Referenced by db\_copy(), db\_create\_key(), db\_delete\_key1(), db\_open\_database(), db\_set\_data(), db\_set\_data\_index(), db\_set\_record(), and db\_set\_value().

#### **7.30.1.11 DWORD KEY::type**

TID\_xxx type

Definition at line 1204 of file midas.h.

Referenced by cm\_transition(), db\_check\_record(), db\_copy(), db\_create\_key(), db\_delete\_key1(), db\_enum\_key(), db\_find\_key(), db\_get\_data(), db\_get\_data\_index(), db\_get\_key(), db\_get\_key\_info(), db\_get\_record(), db\_get\_record\_size(), db\_get\_value(), db\_open\_database(), db\_paste(), db\_set\_data(), db\_set\_data\_index(), db\_set\_record(), db\_set\_value(), main(), and update\_odb().

## 7.31 KEYLIST Struct Reference

### Data Fields

- INT **parent**
- INT **num\_keys**
- INT **first\_key**

### 7.31.1 Field Documentation

#### 7.31.1.1 INT KEYLIST::first\_key

Address of first key

Definition at line 1220 of file midas.h.

Referenced by db\_create\_key(), db\_delete\_key1(), db\_enum\_key(), db\_find\_key(), and db\_open\_database().

#### 7.31.1.2 INT KEYLIST::num\_keys

number of keys

Definition at line 1219 of file midas.h.

Referenced by db\_create\_key(), db\_delete\_key1(), db\_enum\_key(), db\_find\_key(), db\_get\_key\_info(), and db\_open\_database().

#### 7.31.1.3 INT KEYLIST::parent

Address of parent key

Definition at line 1218 of file midas.h.

Referenced by db\_create\_key(), db\_delete\_key1(), db\_enum\_key(), db\_find\_key(), and db\_open\_database().

## 7.32 OPEN\_RECORD Struct Reference

### Data Fields

- INT handle
- WORD access\_mode
- WORD flags

### 7.32.1 Field Documentation

#### 7.32.1.1 WORD OPEN\_RECORD::access\_mode

R/W flags

Definition at line 442 of file msystem.h.

Referenced by cm\_cleanup().

#### 7.32.1.2 WORD OPEN\_RECORD::flags

Data format, ...

Definition at line 443 of file msystem.h.

#### 7.32.1.3 INT OPEN\_RECORD::handle

Handle of record base key

Definition at line 441 of file msystem.h.

Referenced by cm\_cleanup(), and db\_close\_database().



## 7.33 PROGRAM\_INFO Struct Reference

### 7.33.1 Detailed Description

Program information structure

Definition at line 1519 of file midas.h.

### 7.33.2 Field Documentation

#### 7.33.2.1 char PROGRAM\_INFO::alarm\_class[32]

Definition at line 1527 of file midas.h.

#### 7.33.2.2 BOOL PROGRAM\_INFO::auto\_restart

Definition at line 1526 of file midas.h.

#### 7.33.2.3 BOOL PROGRAM\_INFO::auto\_start

Definition at line 1524 of file midas.h.

Referenced by cm\_transition().

#### 7.33.2.4 BOOL PROGRAM\_INFO::auto\_stop

Definition at line 1525 of file midas.h.

Referenced by cm\_transition().

#### 7.33.2.5 DWORD PROGRAM\_INFO::check\_interval

Definition at line 1522 of file midas.h.

#### 7.33.2.6 DWORD PROGRAM\_INFO::first\_failed

Definition at line 1528 of file midas.h.

#### 7.33.2.7 BOOL PROGRAM\_INFO::required

Definition at line 1520 of file midas.h.

**7.33.2.8 char PROGRAM\_INFO::start\_command[256]**

Definition at line 1523 of file midas.h.

Referenced by cm\_transition().

**7.33.2.9 INT PROGRAM\_INFO::watchdog\_timeout**

Definition at line 1521 of file midas.h.

## 7.34 RECORD\_LIST Struct Reference

### 7.34.1 Field Documentation

#### 7.34.1.1 WORD RECORD\_LIST::access\_mode

Definition at line 497 of file msystem.h.

Referenced by db\_close\_all\_records(), db\_open\_record(), db\_send\_changed\_records(), and db\_update\_record().

#### 7.34.1.2 INT RECORD\_LIST::buf\_size

Definition at line 500 of file msystem.h.

Referenced by db\_open\_record(), and db\_update\_record().

#### 7.34.1.3 void\* RECORD\_LIST::copy

Definition at line 499 of file msystem.h.

Referenced by db\_open\_record(), and db\_send\_changed\_records().

#### 7.34.1.4 void\* RECORD\_LIST::data

Definition at line 498 of file msystem.h.

Referenced by db\_open\_record().

#### 7.34.1.5 void(\* RECORD\_LIST::dispatcher)(INT,INT,void\*)

Referenced by db\_open\_record().

#### 7.34.1.6 HANDLE RECORD\_LIST::handle

Definition at line 495 of file msystem.h.

Referenced by db\_close\_all\_records(), db\_close\_record(), db\_open\_record(), and db\_update\_record().

#### 7.34.1.7 HANDLE RECORD\_LIST::hDB

Definition at line 496 of file msystem.h.

Referenced by db\_close\_record(), and db\_open\_record().

**7.34.1.8 void\* RECORD\_LIST::info**

Definition at line 502 of file msystem.h.

Referenced by db\_open\_record().

## 7.35 REQUEST\_LIST Struct Reference

### 7.35.1 Field Documentation

#### 7.35.1.1 INT REQUEST\_LIST::buffer\_handle

Definition at line 509 of file msystem.h.

Referenced by `bm_close_buffer()`, `bm_push_event()`, and `bm_request_event()`.

#### 7.35.1.2 void(\* REQUEST\_LIST::dispatcher)(HANDLE,HANDLE,EVENT\_HEADER\*,void\*)

Referenced by `bm_push_event()`, and `bm_request_event()`.

#### 7.35.1.3 short int REQUEST\_LIST::event\_id

Definition at line 510 of file msystem.h.

Referenced by `bm_request_event()`.

#### 7.35.1.4 short int REQUEST\_LIST::trigger\_mask

Definition at line 511 of file msystem.h.

Referenced by `bm_request_event()`.

## 7.36 RUNINFO Struct Reference

### 7.36.1 Detailed Description

Contains the main parameters regarding the run status. The containt reflects the current system ONLY if Midas clients are connected. Otherwise the status is erroneous.

Definition at line 1482 of file midas.h.

#### Data Fields

- INT **state**
- INT **online\_mode**
- INT **run\_number**
- INT **transition\_in\_progress**
- INT **requested\_transition**
- char **start\_time** [32]
- DWORD **start\_time\_binary**
- char **stop\_time** [32]
- DWORD **stop\_time\_binary**

### 7.36.2 Field Documentation

#### 7.36.2.1 INT RUNINFO::online\_mode

Mode of operation online/offline

Definition at line 1484 of file midas.h.

#### 7.36.2.2 INT RUNINFO::requested\_transition

Deferred transition request

Definition at line 1487 of file midas.h.

#### 7.36.2.3 INT RUNINFO::run\_number

Current processing run number

Definition at line 1485 of file midas.h.

**7.36.2.4 char RUNINFO::start\_time[32]**

ASCII of the last start time

Definition at line 1488 of file midas.h.

**7.36.2.5 DWORD RUNINFO::start\_time\_binary**

Bin of the last start time

Definition at line 1489 of file midas.h.

**7.36.2.6 INT RUNINFO::state**

Current run condition

Definition at line 1483 of file midas.h.

**7.36.2.7 char RUNINFO::stop\_time[32]**

ASCII of the last stop time

Definition at line 1490 of file midas.h.

**7.36.2.8 DWORD RUNINFO::stop\_time\_binary**

ASCII of the last stop time

Definition at line 1491 of file midas.h.

**7.36.2.9 INT RUNINFO::transition\_in\_progress**

Intermediate state during transition

Definition at line 1486 of file midas.h.

## 7.37 TAG Struct Reference

### Data Fields

- char **name** [NAME\_LENGTH]
- DWORD **type**
- DWORD **n\_data**

### 7.37.1 Field Documentation

#### 7.37.1.1 DWORD TAG::n\_data

- 

Definition at line 1333 of file midas.h.

#### 7.37.1.2 char TAG::name[NAME\_LENGTH]

- 

Definition at line 1331 of file midas.h.

#### 7.37.1.3 DWORD TAG::type

- 

Definition at line 1332 of file midas.h.



## Chapter 8

# Midas File Documentation

### 8.1 analyzer.dox File Reference

## 8.2 appendixA.dox File Reference

## 8.3 appendixB.dox File Reference

## 8.4 appendixC.dox File Reference

## 8.5 appendixD.dox File Reference

## 8.6 appendixE.dox File Reference

## 8.7 appendixG.dox File Reference

## 8.8 components.dox File Reference



## 8.9 ebuser.c File Reference

### Functions

- INT **eb\_begin\_of\_run** (INT, char \*, char \*)
- INT **eb\_end\_of\_run** (INT, char \*)
- INT **eb\_user** (INT nfrag, EBUILDER\_CHANNEL \***ebch**, EVENT\_HEADER \***pheader**, void \***pevent**, INT \***dest\_size**)

### 8.9.1 Function Documentation

#### 8.9.1.1 INT eb\_begin\_of\_run (INT, char \*, char \*)

The Event builder user file.

file **ebuser.c**(p.??)

Referenced by **tr\_prestart()**.

#### 8.9.1.2 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 **main()**.

#### 8.9.1.3 INT eb\_user (INT *nfrag*, EBUILDER\_CHANNEL \**ebch*, EVENT\_HEADER \**pheader*, void \**pevent*, INT \**dest\_size*)

Hook to the event builder task after the reception of all fragments of the same serial number. The destination event has already the final **EVENT\_HEADER**(p.??) setup with the data size set to 0. It is then 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**(p.??) 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.
- ebch* Structure to all the fragments.
- pheader* Destination pointer to the header.
- pevent* Destination pointer to the bank header.
- dest\_size* Destination event size in bytes.

#### Returns:

EB\_SUCCESS

Definition at line 137 of file ebuser.c.

Referenced by source\_scan().

**8.9.1.4** `INT ebuser (INT, EBUILDER_CHANNEL *,  
EVENT_HEADER *, void *, INT *)`

## **8.9.2 Variable Documentation**

**8.9.2.1** `INT lModulo = 100`

Global var for testing.

Definition at line 45 of file ebuser.c.

Referenced by `eb_begin_of_run()`, and `eb_user()`.

## 8.10 esone.c File Reference

### Functions

- `INLINE void ccinit (void)`
- `INLINE int fcinit (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 ctc_i (const int ext, int *l)`
- `INLINE void cccd (const int ext, int l)`
- `INLINE void cted (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[])`

### 8.10.1 Define Documentation

#### 8.10.1.1 #define INLINE

Definition at line 34 of file esone.c.

Referenced by `came_cn()`, `came_ext()`, `cccc()`, `cccd()`, `ccci()`, `cccz()`, `ccinit()`, `cclc()`, `cclm()`, `cclnk()`, `ccrgl()`, `cculk()`, `cdlam()`, `cdreg()`, `cfga()`, `cfmad()`, `cfsa()`, `cfubc()`, `cfubr()`, `csga()`, `csmad()`, `cssa()`, `csubc()`, `csubr()`, `ctcd()`, `ctci()`, `ctgl()`, `ctlm()`, and `fcinit()`.

## 8.10.2 Function Documentation

### 8.10.2.1 **INLINE void came\_cn** (int \* *ext*, const int *b*, const int *c*, const int *n*, const int *a*)

Definition at line 40 of file esone.c.

Referenced by cdreg(), cfmad(), and csmad().

### 8.10.2.2 **INLINE void came\_ext** (const int *ext*, int \* *b*, int \* *c*, int \* *n*, int \* *a*)

Definition at line 46 of file esone.c.

Referenced by cccc(), cccd(), ccci(), cc cz(), cclc(), cclm(), cclnk(), ccrgl(), cculk(), cfmad(), cf sa(), csmad(), cssa(), ctcd(), ctci(), ctgl(), and ctlm().

### 8.10.2.3 **INLINE void cccc** (const int *ext*)

cccc Control Crate Clear.

Generate Crate Clear function. Execute **cam\_crate\_clear**(p.??)

Crate Clear.

**Parameters:**

*ext* external address

**Returns:**

void

Definition at line 204 of file esone.c.

### 8.10.2.4 **INLINE void cccd** (const int *ext*, int *l*)

cccd Control Crate D.

Enable or Disable Crate Demand.

Crate D.

**Parameters:**

*ext* external address

*l* action l=0 -> Clear D, l=1 -> Set D

**Returns:**

void

Definition at line 282 of file esone.c.

**8.10.2.5    `INLINE void ccci (const int ext, int l)`**

ccci Control Crate I.

Set or Clear Dataway Inhibit, Execute `cam_inhinit_set()` /`clear()`

Crate I.

**Parameters:**

*ext* external address

*l* action *l*=0 -> Clear I, *l*=1 -> Set I

**Returns:**

void

Definition at line 241 of file `esone.c`.

**8.10.2.6    `INLINE void cccz (const int ext)`**

cccz Control Crate Z.

Generate Dataway Initialize. Execute `cam_crate_zinit()`(p.??)

Crate Z.

**Parameters:**

*ext* external address

**Returns:**

void

Definition at line 222 of file `esone.c`.

**8.10.2.7    `INLINE void ccinit (void)`**

ccinit CAMAC initialization

CAMAC initialization must be called before any other ESONE subroutine call

CAMAC initialization.

**Returns:**

void

Definition at line 68 of file `esone.c`.

#### 8.10.2.8 **INLINE void cclc (const int *lam*)**

cclc Control Clear LAM. Clear the LAM of the station pointer by the lam address.

Clear LAM.

**Parameters:**

*lam* external address

**Returns:**

void

Definition at line 450 of file esone.c.

#### 8.10.2.9 **INLINE void cclm (const int *lam*, int *l*)**

cclm Control Crate LAM.

Enable or Disable LAM. Execute F24 for disable, F26 for enable.

dis/enable LAM.

**Parameters:**

*lam* external address

*l* action l=0 -> disable LAM , l=1 -> enable LAM

**Returns:**

void

Definition at line 367 of file esone.c.

#### 8.10.2.10 **INLINE void cclnk (const int *lam*, void(\* *isr*)(void))**

cclnk Link LAM to service procedure

Link a specific service routine to a LAM. Since this routine is executed asynchronously, care must be taken on re-entrancy.

Link LAM to service procedure

**Parameters:**

*lam* external address

*isr* name of service procedure

**Returns:**

void

Definition at line 391 of file esone.c.

**8.10.2.11** **INLINE void ccrgl (const int *lam*)**

ccrgl Relink LAM

Re-enable LAM in the controller

Re-enable LAM

**Parameters:**

*lam* external address

**Returns:**

void

Definition at line 430 of file esone.c.

**8.10.2.12** **INLINE void cculk (const int *lam*)**

cculk Unlink LAM from service procedure

Performs complementary operation to cclnk.

Detach LAM from service procedure

**Parameters:**

*lam* external address

**Returns:**

void

Definition at line 412 of file esone.c.

**8.10.2.13** **INLINE void cdlam (int \* *lam*, const int *b*, const int *c*,  
const int *n*, const int *a*, const int *inta*[2])**

cdlam Control Declare LAM.

Declare LAM, Identical to cdreg.

Declare LAM.

**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[2]* implementation dependent

**Returns:**

void

Definition at line 328 of file esone.c.

**8.10.2.14** **INLINE void cdreg (int \* *ext*, const int *b*, const int *c*,  
const int *n*, const int *a*)**

cdreg 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 {appendix D: Supported hardware}. In this case External Address register.

**Parameters:**

*ext* external address

*b* branch number (0..7)

*c* crate number (0..)

*n* station number (0..30)

*a* sub-address (0..15)

**Returns:**

void

Definition at line 107 of file esone.c.

Referenced by cdlam().

**8.10.2.15** **INLINE void cfga (int *f*[], int *exta*[], int *intc*[], int *qa*[],  
int *cb*[])**

cfga Control Full (24bit) word General Action.

General external address scan function.

**Parameters:**

*f* function code

*exta*[] external address array

*intc*[] data array

*qa*[] Q response array

*cb[]* control block array \ *cb[0]* : number of function to perform \ *cb[1]* :  
returned number of function performed

**Returns:**

void

Definition at line 491 of file esone.c.

**8.10.2.16    `INLINE void cfmad (int f, int extb[], int intc[], int cb[])`**

cfmad 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.}

Address scan function.

**Parameters:**

*f* function code

*extb[]* external address array \ *extb[0]* : first valid external address \  
*extb[1]* : last valid external address

*intc[]* data array

*qa[]* Q response array

*cb[]* control block array \ *cb[0]* : number of function to perform \ *cb[1]* :  
returned number of function performed

**Returns:**

void

Definition at line 548 of file esone.c.

**8.10.2.17    `INLINE void cfsa (const int f, const int ext, unsigned long * d, int * q)`**

cfsa 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)

24 bit function.

**Parameters:**

*f* function code (0..31)

*ext* external address

*d* data long word

*q* Q response

**Returns:**

void

Definition at line 170 of file esone.c.

Referenced by cfga(), cfubc(), and cfubr().

**8.10.2.18** `INLINE void cfubc (const int f, int ext, int intc[], int cb[])`

cfubc Control Full (24bit) Block Repeat with Q-stop.

Execute function *f* on address *ext* with data *intc*[] while Q.

Repeat function Q-stop.

**Parameters:**

*f* function code

*ext* external address array

*intc*[] data array

*cb*[] control block array \ *cb*[0] : number of function to perform \ *cb*[1] : returned number of function performed

**Returns:**

void

Definition at line 656 of file esone.c.

**8.10.2.19** `INLINE void cfubr (const int f, int ext, int intc[], int cb[])`

cfubr 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.

Repeat function.

**Parameters:**

*f* function code

*ext* external address array

*intc*[] data array

*cb*[] control block array \ *cb*[0] : number of function to perform \ *cb*[1] :  
returned number of function performed

**Returns:**

void

Definition at line 725 of file esone.c.

**8.10.2.20** **INLINE void csga (int *f*[], int *exta*[], int *intc*[], int *qa*[],  
int *cb*[])**

csga Control (16bit) word General Action.

General external address scan function.

**Parameters:**

*f* function code

*exta*[] external address array

*intc*[] data array

*qa*[] Q response array

*cb*[] control block array \ *cb*[0] : number of function to perform \ *cb*[1] :  
returned number of function performed

**Returns:**

void

Definition at line 515 of file esone.c.

**8.10.2.21** **INLINE void csmad (int *f*, int *extb*[], int *intc*[], int *cb*[])**

csmad 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.}

Address scan function.

**Parameters:**

*f* function code  
*extb[]* external address array \ extb[0] : first valid external address \  
extb[1] : last valid external address  
*intc[]* data array  
*qa[]* Q response array  
*cb[]* control block array \ cb[0] : number of function to perform \ cb[1] :  
returned number of function performed

**Returns:**

void

Definition at line 606 of file esone.c.

#### 8.10.2.22 **INLINE void cssa (const int *f*, int *ext*, unsigned short \* *d*, int \* *q*)**

cssa 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)

16 bit function.

**Parameters:**

*f* function code (0..31)  
*ext* external address  
*d* data word  
*q* Q response

**Returns:**

void

Definition at line 129 of file esone.c.

Referenced by csga(), csubc(), and csubr().

#### 8.10.2.23 **INLINE void csubc (const int *f*, int *ext*, int *intc*[], int *cb*[])**

csubc Control (16bit) Block Repeat with Q-stop.

Execute function *f* on address *ext* with data *intc*[] while *Q*.

Repeat function *Q*-stop.

**Parameters:**

*f* function code

*ext* external address array

*intc*[] data array

*cb*[] control block array \ *cb*[0] : number of function to perform \ *cb*[1] :  
returned number of function performed

**Returns:**

void

Definition at line 690 of file *esone.c*.

#### 8.10.2.24 **INLINE void csubr (const int *f*, int *ext*, int *intc*[], int *cb*[])**

*csubr* Repeat Mode Block Transfer (16bit).

Execute function *f* on address *ext* with data *intc*[] if *Q*. If no *Q* keep current *intc*[] data. Repeat *cb*[0] times.

Repeat function.

**Parameters:**

*f* function code

*ext* external address array

*intc*[] data array

*cb*[] control block array \ *cb*[0] : number of function to perform \ *cb*[1] :  
returned number of function performed

**Returns:**

void

Definition at line 759 of file *esone.c*.

#### 8.10.2.25 **INLINE void ctcd (const int *ext*, int \* *l*)**

*ctcd* Control Test Crate D.

Test Crate Demand.

Test Crate D.

**Parameters:**

*ext* external address

*l* D cleared -> l=0, D set -> l=1

**Returns:**

void

Definition at line 305 of file esone.c.

**8.10.2.26** `INLINE void ctci (const int ext, int * l)`

ctci Test Crate I.

Test Crate Inhibit, Execute `cam__inhibit__test()`(p.??)

Crate I.

**Parameters:**

*ext* external address

*l* action l=0 -> Clear I, l=1 -> Set I

**Returns:**

void

Definition at line 263 of file esone.c.

**8.10.2.27** `INLINE void ctgl (const int ext, int * l)`

ctgl Control Test Demand Present.

Test the LAM register.

Test GL.

**Parameters:**

*lam* external LAM register address

*l* l !=0 if any LAM is set.

**Returns:**

void

Definition at line 346 of file esone.c.

**8.10.2.28** `INLINE void ctlm (const int lam, int * l)`

ctlm Test LAM.

Test the LAM of the station pointed by lam. Performs an F8

Test LAM.

**Parameters:**

*lam* external address

*l* No LAM-> l=0, LAM present-> l=1

**Returns:**

void

Definition at line 469 of file esone.c.

**8.10.2.29** `INLINE int fccinit (void)`

fccinit CAMAC initialization with return status

fccinit can be called instead of ccinit to determine if the initialization was successful

CAMAC initialization.

**Returns:**

1 for success, 0 for failure

Definition at line 83 of file esone.c.



## 8.11 eventbuilder.dox File Reference

## 8.12 internal.dox File Reference

## 8.13 introduction.dox File Reference

## 8.14 mcstd.h File Reference

### 8.14.1 Detailed Description

The Midas CAMAC include file

Definition in file **mcstd.h**.

### Functions

- EXTERNAL INLINE void EXPRT **cam16i** (const int c, const int n, const int a, const int f, WORD \*d)
- EXTERNAL INLINE void EXPRT **cam24i** (const int c, const int n, const int a, const int f, DWORD \*d)
- EXTERNAL INLINE void EXPRT **cam8i\_q** (const int c, const int n, const int a, const int f, BYTE \*d, int \*x, int \*q)
- EXTERNAL INLINE void EXPRT **cam16i\_q** (const int c, const int n, const int a, const int f, WORD \*d, int \*x, int \*q)
- EXTERNAL INLINE void EXPRT **cam24i\_q** (const int c, const int n, const int a, const int f, DWORD \*d, int \*x, int \*q)
- EXTERNAL INLINE void EXPRT **cam16i\_r** (const int c, const int n, const int a, const int f, WORD \*\*d, const int r)
- EXTERNAL INLINE void EXPRT **cam24i\_r** (const int c, const int n, const int a, const int f, DWORD \*\*d, const int r)
- EXTERNAL INLINE void EXPRT **cam8i\_rq** (const int c, const int n, const int a, const int f, BYTE \*\*d, const int r)
- EXTERNAL INLINE void EXPRT **cam16i\_rq** (const int c, const int n, const int a, const int f, WORD \*\*d, const int r)
- EXTERNAL INLINE void EXPRT **cam24i\_rq** (const int c, const int n, const int a, const int f, DWORD \*\*d, const int r)
- EXTERNAL INLINE void EXPRT **cam8i\_sa** (const int c, const int n, const int a, const int f, BYTE \*\*d, const int r)
- EXTERNAL INLINE void EXPRT **cam16i\_sa** (const int c, const int n, const int a, const int f, WORD \*\*d, const int r)
- EXTERNAL INLINE void EXPRT **cam24i\_sa** (const int c, const int n, const int a, const int f, DWORD \*\*d, const int r)
- EXTERNAL INLINE void EXPRT **cam8i\_sn** (const int c, const int n, const int a, const int f, BYTE \*\*d, const int r)
- EXTERNAL INLINE void EXPRT **cam16i\_sn** (const int c, const int n, const int a, const int f, WORD \*\*d, const int r)
- EXTERNAL INLINE void EXPRT **cam24i\_sn** (const int c, const int n, const int a, const int f, DWORD \*\*d, const int r)
- EXTERNAL INLINE void EXPRT **cami** (const int c, const int n, const int a, const int f, WORD \*d)

- EXTERNAL INLINE void EXPRT **cam8o** (const int c, const int n, const int a, const int f, BYTE d)
- EXTERNAL INLINE void EXPRT **cam16o** (const int c, const int n, const int a, const int f, WORD d)
- EXTERNAL INLINE void EXPRT **cam24o** (const int c, const int n, const int a, const int f, DWORD d)
- EXTERNAL INLINE void EXPRT **cam8o\_q** (const int c, const int n, const int a, const int f, BYTE d, int \*x, int \*q)
- EXTERNAL INLINE void EXPRT **cam16o\_q** (const int c, const int n, const int a, const int f, WORD d, int \*x, int \*q)
- EXTERNAL INLINE void EXPRT **cam24o\_q** (const int c, const int n, const int a, const int f, DWORD d, int \*x, int \*q)
- EXTERNAL INLINE void EXPRT **cam8o\_r** (const int c, const int n, const int a, const int f, BYTE \*d, const int r)
- EXTERNAL INLINE void EXPRT **cam16o\_r** (const int c, const int n, const int a, const int f, WORD \*d, const int r)
- EXTERNAL INLINE void EXPRT **cam24o\_r** (const int c, const int n, const int a, const int f, DWORD \*d, const int r)
- EXTERNAL INLINE void EXPRT **camo** (const int c, const int n, const int a, const int f, WORD d)
- EXTERNAL INLINE int EXPRT **camc\_chk** (const int c)
- EXTERNAL INLINE void EXPRT **camc** (const int c, const int n, const int a, const int f)
- EXTERNAL INLINE void EXPRT **camc\_q** (const int c, const int n, const int a, const int f, int \*q)
- EXTERNAL INLINE void EXPRT **camc\_sa** (const int c, const int n, const int a, const int f, const int r)
- EXTERNAL INLINE void EXPRT **camc\_sn** (const int c, const int n, const int a, const int f, const int r)
- EXTERNAL INLINE int EXPRT **cam\_init** (void)
- EXTERNAL INLINE int EXPRT **cam\_init\_rpc** (char \*host\_name, char \*exp\_name, char \*fe\_name, char \*client\_name, char \*rpc\_server)
- EXTERNAL INLINE void EXPRT **cam\_exit** (void)
- EXTERNAL INLINE void EXPRT **cam\_inhibit\_set** (const int c)
- EXTERNAL INLINE void EXPRT **cam\_inhibit\_clear** (const int c)
- EXTERNAL INLINE int EXPRT **cam\_inhibit\_test** (const int c)
- EXTERNAL INLINE void EXPRT **cam\_crate\_clear** (const int c)
- EXTERNAL INLINE void EXPRT **cam\_crate\_zinit** (const int c)
- EXTERNAL INLINE void EXPRT **cam\_lam\_enable** (const int c, const int n)
- EXTERNAL INLINE void EXPRT **cam\_lam\_disable** (const int c, const int n)
- EXTERNAL 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)

## 8.15 mevb.c File Reference

### Functions

- INT **eb\_mfragment\_add** (char \*pdest, char \*psrce, INT \*size)
- INT **eb\_begin\_of\_run** (INT, char \*, char \*)
- INT **eb\_end\_of\_run** (INT, char \*)
- INT **eb\_user** (INT, EBUILDER\_CHANNEL \*, **EVENT\_HEADER** \*, void \*, INT \*)
- INT **source\_scan** (INT fmt, INT nfragment, HANDLE dest\_hBuf, char \*dest\_event)

### Variables

- EBUILDER\_SETTINGS **ebset**

#### 8.15.1 Function Documentation

##### 8.15.1.1 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

Definition at line 55 of file ebuser.c.

##### 8.15.1.2 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

Definition at line 70 of file ebuser.c.

**8.15.1.3 INT eb\_mfragment\_add (char \* *pdest*, char \* *psrce*, INT \* *size*)****Parameters:***pdest* Destination pointer*psrce* Fragment source pointer*size* Current destination event size (byte)**Returns:**

EB\_SUCCESS

Definition at line 84 of file mevb.c.

Referenced by main().

**8.15.1.4 INT eb\_user (INT *nfrag*, EBUILDER\_CHANNEL \* *ebch*, EVENT\_HEADER \* *pheader*, void \* *pevent*, INT \* *dest\_size*)**

Hook to the event builder task after the reception of all fragments of the same serial number. The destination event has already the final **EVENT\_HEADER**(p.??) 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**(p.??) 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.  
*ebch* Structure to all the fragments.  
*pheader* Destination pointer to the header.  
*pevent* Destination pointer to the bank header.  
*dest\_size* Destination event size in bytes.

#### Returns:

EB\_SUCCESS

Definition at line 137 of file ebuser.c.

Referenced by source\_scan().

#### 8.15.1.5 INT eb\_yfragment\_add (char \* *pdest*, char \* *psrce*, INT \* *size*)

Definition at line 141 of file mevb.c.

Referenced by main().

#### 8.15.1.6 void free\_event\_buffer (INT *nfrag*)

Definition at line 283 of file mevb.c.

Referenced by main(), source\_booking(), and source\_unbooking().

**8.15.1.7 INT handFlush (INT)**

Definition at line 296 of file mevb.c.

Referenced by main().

**8.15.1.8 int main (unsigned int *argc*, char \*\* *argv*)**

Definition at line 635 of file mevb.c.

**8.15.1.9 INT source\_booking (INT *nfrag*)**

Definition at line 329 of file mevb.c.

Referenced by tr\_prestart().

**8.15.1.10 INT source\_scan (INT *fmt*, INT *nfragment*, HANDLE *dest\_hBuf*, char \* *dest\_event*)**

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 collected, 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  
*nfragment* number of fragment to collect  
*dest\_hBuf* Destination buffer handle  
*dest\_event* destination point for built event

**Returns:**

EB\_NO\_MORE\_EVENT, EB\_COMPOSE\_TIMEOUT if different then  
SUCCESS (bm\_compose, rpc\_sent error)

Definition at line 470 of file mevb.c.

Referenced by `main()`.

#### 8.15.1.11 INT `source_unbooking (nfrag)`

Definition at line 413 of file mevb.c.

Referenced by `main()`.

#### 8.15.1.12 INT `tr_prestart (INT rn, char * error)`

Definition at line 216 of file mevb.c.

Referenced by `main()`.

#### 8.15.1.13 INT `tr_stop (INT rn, char * error)`

Definition at line 270 of file mevb.c.

Referenced by `main()`.

#### 8.15.1.14 INT `ybos_event_swap (DWORD * pevt)`

Referenced by `eb_yfragment_add()`, and `source_scan()`.

### 8.15.2 Variable Documentation

#### 8.15.2.1 **BOOL** `abort_requested = FALSE` `stop_requested = TRUE`

Definition at line 56 of file mevb.c.

Referenced by `main()`, `source_scan()`, and `tr_prestart()`.

#### 8.15.2.2 **DWORD** `cdemask = 0`

Definition at line 62 of file mevb.c.

Referenced by `main()`, `source_scan()`, and `tr_prestart()`.

#### 8.15.2.3 **BOOL** `debug = FALSE` `debug1=FALSE`

Definition at line 54 of file mevb.c.

**8.15.2.4 EUILDER\_CHANNEL ebch[MAX\_CHANNELS]**

Definition at line 49 of file mevb.c.

Referenced by eb\_user(), free\_event\_buffer(), handFlush(), main(), source\_booking(), source\_scan(), source\_unbooking(), and tr\_prestart().

**8.15.2.5 EUILDER\_SETTINGS ebset**

The Event builder main file.

file **mevb.c**(p.??)

Definition at line 47 of file mevb.c.

Referenced by main(), source\_booking(), source\_scan(), and tr\_prestart().

**8.15.2.6 EUILDER\_STATISTICS ebstat**

Definition at line 48 of file mevb.c.

Referenced by main(), source\_scan(), and tr\_prestart().

**8.15.2.7 DWORD gbl\_bytes\_sent = 0 gbl\_events\_sent=0**

Definition at line 61 of file mevb.c.

Referenced by main(), source\_scan(), and tr\_prestart().

**8.15.2.8 INT gbl\_run = 0**

Definition at line 63 of file mevb.c.

Referenced by main(), and tr\_prestart().

**8.15.2.9 HANDLE hDB**

Definition at line 53 of file mevb.c.

**8.15.2.10 HANDLE hKey**

Definition at line 53 of file mevb.c.

Referenced by 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\_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\_find\_key(), db\_get\_data(), db\_get\_data\_index(), db\_get\_key(), db\_get\_key\_info(), db\_get\_key\_time(), db\_get\_record(), db\_get\_record\_size(), db\_open\_record(), db\_paste(), db\_save(), db\_save\_struct(), db\_set\_data(), db\_set\_data\_index(), db\_set\_record(), db\_set\_value(), db\_update\_record(), logger\_root(), main(), and update\_odb().

#### 8.15.2.11 HANDLE hStatKey

Definition at line 53 of file mevb.c.

Referenced by main(), and tr\_prestart().

#### 8.15.2.12 DWORD max\_event\_size = MAX\_EVENT\_SIZE

Definition at line 51 of file mevb.c.

Referenced by handFlush(), main(), scheduler(), send\_event(), source\_booking(), and source\_scan().

#### 8.15.2.13 INT(\* meb\_fragment\_add)(char \*, char \*, INT \*)

Definition at line 65 of file mevb.c.

Referenced by main(), and source\_scan().

#### 8.15.2.14 INT run\_state = 0

Definition at line 59 of file mevb.c.

#### 8.15.2.15 DWORD start\_time = 0 stop\_time=0 request\_stop\_time=0

Definition at line 60 of file mevb.c.

Referenced by bm\_check\_buffers(), cm\_shutdown(), main(), and tr\_prestart().

#### 8.15.2.16 BOOL stopped = TRUE

Definition at line 57 of file mevb.c.

Referenced by handFlush(), and tr\_prestart().

#### **8.15.2.17    BOOL wheel = FALSE**

Definition at line 58 of file mevb.c.

Referenced by main(), and source\_scan().

## 8.16 mfe.c File Reference

### 8.16.1 Define Documentation

#### 8.16.1.1 `#define DEFAULT_FE_TIMEOUT 60000`

Definition at line 265 of file mfe.c.

Referenced by `main()`.

#### 8.16.1.2 `#define EQUIPMENT_COMMON_STR "\Event ID = WORD: 0 \ n \ Trigger mask = WORD: 0 \ n \ Buffer = STRING: [32] SYSTEM \ n \ Type = INT: 0 \ n \ Source = INT: 0 \ n \ Format = STRING: [8] FIXED \ n \ Enabled = BOOL: 0 \ n \ Read on = INT: 0 \ n \ Period = INT: 0 \ n \ Event limit = DOUBLE: 0 \ n \ Num subevents = DWORD: 0 \ n \ Log history = INT: 0 \ n \ Frontend host = STRING: [32] \ n \ Frontend name = STRING: [32] \ n \ Frontend file name = STRING: [256] \ n \ "`

Definition at line 349 of file mfe.c.

#### 8.16.1.3 `#define EQUIPMENT_STATISTICS_STR " \ Events sent = DOUBLE: 0 \ n \ Events per sec. = DOUBLE: 0 \ n \ kBytes per sec. = DOUBLE: 0 \ n \ "`

Definition at line 367 of file mfe.c.

#### 8.16.1.4 `#define ODB_UPDATE_TIME 1000`

Definition at line 263 of file mfe.c.

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

#### 8.16.1.5 `#define SERVER_CACHE_SIZE 100000`

Definition at line 261 of file mfe.c.

Referenced by `scheduler()`.

## 8.16.2 Function Documentation

**8.16.2.1** INT `begin_of_run` (INT *run\_number*, char \* *error*)

**8.16.2.2** void `display` (BOOL *bInit*)

Definition at line 1874 of file mfe.c.

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

**8.16.2.3** INT `end_of_run` (INT *run\_number*, char \* *error*)

**8.16.2.4** INT `frontend_exit` (void)

Referenced by `main()`.

**8.16.2.5** INT `frontend_init` (void)

Referenced by `main()`.

**8.16.2.6** INT `frontend_loop` (void)

Referenced by `scheduler()`.

**8.16.2.7** INT `interrupt_configure` (INT *cmd*, INT *source*, PTYPE *adr*)

Referenced by `interrupt_enable()`, and `main()`.

**8.16.2.8** void `interrupt_enable` (BOOL *flag*)

Definition at line 1725 of file mfe.c.

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

**8.16.2.9** void `interrupt_routine` (void)

Definition at line 1751 of file mfe.c.

**8.16.2.10** BOOL `logger_root` ()

Definition at line 2012 of file mfe.c.



Referenced by scheduler().

#### 8.16.2.11 int main (int *argc*, char \* *argv*[])

Definition at line 3138 of file mfe.c.

#### 8.16.2.12 INT manual\_trigger (INT *index*, void \* *prpc\_param*[])

Definition at line 508 of file mfe.c.

#### 8.16.2.13 int message\_print (const char \* *msg*)

Definition at line 1847 of file mfe.c.

Referenced by main().

#### 8.16.2.14 INT pause\_run (INT *run\_number*, char \* *error*)

#### 8.16.2.15 INT poll\_event (INT *source*, INT *count*, BOOL *test*)

Referenced by scheduler().

#### 8.16.2.16 INT register\_equipment (void)

Definition at line 526 of file mfe.c.

Referenced by main().

#### 8.16.2.17 INT resume\_run (INT *run\_number*, char \* *error*)

#### 8.16.2.18 INT scheduler (void)

Definition at line 2063 of file mfe.c.

Referenced by main().

#### 8.16.2.19 void send\_all\_periodic\_events (INT *transition*)

Definition at line 1675 of file mfe.c.

**8.16.2.20 int send\_event (INT *index*)**

Definition at line 1273 of file mfe.c.

Referenced by scheduler(), and send\_all\_periodic\_events().

**8.16.2.21 INT tr\_prepause (INT *rn*, char \* *error*)**

Definition at line 456 of file mfe.c.

Referenced by main().

**8.16.2.22 INT tr\_prestop (INT *rn*, char \* *error*)**

Definition at line 414 of file mfe.c.

Referenced by main(), and tr\_prestop().

**8.16.2.23 INT tr\_resume (INT *rn*, char \* *error*)**

Definition at line 483 of file mfe.c.

Referenced by main().

**8.16.2.24 INT tr\_start (INT *rn*, char \* *error*)**

Definition at line 377 of file mfe.c.

Referenced by main().

**8.16.2.25 void update\_odb (EVENT\_HEADER \* *pevent*, HANDLE *hKey*, INT *format*)**

Definition at line 969 of file mfe.c.

Referenced by scheduler(), and send\_event().

**8.16.3 Variable Documentation****8.16.3.1 DWORD actual\_millitime**

Definition at line 273 of file mfe.c.

Referenced by interrupt\_routine(), main(), and scheduler().

**8.16.3.2   DWORD actual\_time**

Definition at line 271 of file mfe.c.

Referenced by interrupt\_routine(), and scheduler().

**8.16.3.3   DWORD auto\_restart = 0**

Definition at line 289 of file mfe.c.

Referenced by scheduler().

**8.16.3.4   BOOL debug**

Definition at line 287 of file mfe.c.

Referenced by handFlush(), main(), source\_booking(), source\_scan(), and source\_unbooking().

**8.16.3.5   INT display\_period**

Definition at line 234 of file mfe.c.

Referenced by main(), and scheduler().

**8.16.3.6   EQUIPMENT equipment[]**

Definition at line 326 of file mfe.c.

Referenced by display(), main(), register\_equipment(), scheduler(), send\_all\_periodic\_events(), and send\_event().

**8.16.3.7   INT event\_buffer\_size**

Definition at line 232 of file mfe.c.

Referenced by main().

**8.16.3.8   char exp\_name[NAME\_LENGTH]**

Definition at line 278 of file mfe.c.

Referenced by cm\_connect\_experiment(), cm\_connect\_experiment1(), cm\_get\_environment(), cm\_list\_experiments(), cm\_select\_experiment(), and main().

**8.16.3.9 INT fe\_stop = 0**

Definition at line 285 of file mfe.c.

Referenced by scheduler().

**8.16.3.10 BOOL frontend\_call\_loop**

Definition at line 226 of file mfe.c.

Referenced by scheduler().

**8.16.3.11 char\* frontend\_file\_name**

Definition at line 224 of file mfe.c.

**8.16.3.12 char\* frontend\_name**

Definition at line 222 of file mfe.c.

Referenced by display(), and main().

**8.16.3.13 HANDLE hDB**

Definition at line 292 of file mfe.c.

Referenced by al\_trigger\_alarm(), 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\_experiment\_database(), 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\_find\_key(), db\_get\_data(), db\_get\_data\_index(), db\_get\_key(), db\_get\_key\_info(), db\_get\_key\_time(), db\_get\_record(), db\_get\_record\_size(), db\_get\_value(), db\_load(), db\_lock\_database(), db\_open\_database(), db\_open\_record(), db\_paste(), db\_protect\_database(), db\_save(), db\_save\_struct(), 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(), logger\_root(), main(), scheduler(), tr\_prestart(), and update\_odb().

**8.16.3.14 char host\_name[HOST\_NAME\_LENGTH]**

Definition at line 276 of file mfe.c.

Referenced by cm\_connect\_client(), cm\_connect\_experiment(), cm\_connect\_experiment1(), cm\_get\_environment(), cm\_list\_experiments(), cm\_select\_experiment(), cm\_set\_client\_info(), cm\_transition(), display(), and main().

**8.16.3.15 BOOL interrupt\_enabled**

Definition at line 1722 of file mfe.c.

Referenced by interrupt\_enable().

**8.16.3.16 EQUIPMENT\* interrupt\_eq = NULL**

Definition at line 329 of file mfe.c.

Referenced by interrupt\_enable(), interrupt\_routine(), main(), and scheduler().

**8.16.3.17 EVENT\_HEADER\* interrupt\_odb\_buffer**

Definition at line 331 of file mfe.c.

Referenced by interrupt\_routine(), main(), and scheduler().

**8.16.3.18 BOOL interrupt\_odb\_buffer\_valid**

Definition at line 333 of file mfe.c.

Referenced by interrupt\_routine(), and scheduler().

**8.16.3.19 INT max\_bytes\_per\_sec**

Definition at line 281 of file mfe.c.

Referenced by scheduler().

**8.16.3.20 INT max\_event\_size**

Definition at line 228 of file mfe.c.

Referenced by handFlush(), main(), scheduler(), send\_event(), source\_booking(), and source\_scan().

**8.16.3.21 INT max\_event\_size\_frag**

Definition at line 230 of file mfe.c.

Referenced by main(), and send\_event().

**8.16.3.22 INT optimize = 0**

Definition at line 283 of file mfe.c.

Referenced by scheduler().

**8.16.3.23 INT run\_number**

Definition at line 269 of file mfe.c.

Referenced by cm\_transition(), display(), el\_submit(), and scheduler().

**8.16.3.24 INT run\_state**

Definition at line 267 of file mfe.c.

Referenced by display(), handFlush(), main(), scheduler(), and tr\_prestart().

## 8.17 mhttpd.dox File Reference

## 8.18 midas.c File Reference

### 8.18.1 Detailed Description

The main core C-code for Midas.

Definition in file **midas.c**.

### 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 \*filename, INT line, const char \*routine, const char \*format,...)
- **INT cm\_msg1** (INT message\_type, char \*filename, INT line, const char \*facility, const char \*routine, const char \*format,...)
- **INT cm\_msg\_register** (void(\*func)(HANDLE, HANDLE, **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** (HANDLE hDB, INT pid)
- **INT cm\_check\_client** (HANDLE hDB, HANDLE hKeyClient)
- **INT cm\_set\_client\_info** (HANDLE hDB, HANDLE \*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, HANDLE \*hConn)
- INT **cm\_disconnect\_client** (HANDLE hConn, BOOL bShutdown)
- INT **cm\_disconnect\_experiment** (void)
- INT **cm\_set\_experiment\_database** (HANDLE hDB, HANDLE hKeyClient)
- INT **cm\_get\_experiment\_database** (HANDLE \*hDB, HANDLE \*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** (HANDLE hDB, char \*client\_name, DWORD \*timeout, DWORD \*last)
- INT **cm\_register\_transition** (INT transition, INT(\*func)(INT, char \*))
- 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\_flag, INT debug\_flag)
- INT **cm\_yield** (INT millisec)
- INT **cm\_execute** (char \*command, char \*result, INT bufsize)
- INT **bm\_match\_event** (short int event\_id, short int trigger\_mask, **EVENT\_HEADER** \*pevent)
- INT **bm\_open\_buffer** (char \*buffer\_name, INT buffer\_size, INT \*buffer\_handle)
- INT **bm\_close\_buffer** (INT buffer\_handle)
- INT **bm\_close\_all\_buffers** (void)
- INT **cm\_shutdown** (char \*name, BOOL bUnique)
- INT **cm\_exist** (char \*name, BOOL bUnique)
- INT **cm\_cleanup** (char \*client\_name, BOOL ignore\_timeout)
- INT **bm\_set\_cache\_size** (INT buffer\_handle, INT read\_size, INT write\_size)
- INT **bm\_compose\_event** (**EVENT\_HEADER** \*event\_header, short int event\_id, short int trigger\_mask, DWORD size, DWORD serial)
- INT **bm\_request\_event** (HANDLE buffer\_handle, short int event\_id, short int trigger\_mask, INT sampling\_type, HANDLE \*request\_id, void(\*func)(HANDLE, HANDLE, **EVENT\_HEADER** \*, void \*))
- INT **bm\_remove\_event\_request** (INT buffer\_handle, INT request\_id)
- INT **bm\_delete\_request** (INT request\_id)

- **INT** **bm\_send\_event** (INT buffer\_handle, void \*source, INT buf\_size, INT async\_flag)
- **INT** **bm\_flush\_cache** (INT buffer\_handle, INT async\_flag)
- **INT** **bm\_receive\_event** (INT buffer\_handle, void \*destination, INT \*buf\_size, INT async\_flag)
- **INT** **bm\_skip\_event** (INT buffer\_handle)
- **INT** **bm\_push\_event** (char \*buffer\_name)
- **INT** **bm\_check\_buffers** ()
- **INT** **bm\_empty\_buffers** ()
- **INT** **rpc\_register\_client** (char \*name, RPC\_LIST \*list)
- **INT** **rpc\_register\_functions** (RPC\_LIST \*new\_list, INT(\*func)(INT, void \*\*))
- **INT** **rpc\_set\_option** (HANDLE hConn, INT item, INT value)
- **INT** **rpc\_send\_event** (INT buffer\_handle, void \*source, INT buf\_size, INT async\_flag)
- **INT** **rpc\_flush\_event** ()
- **void** **bk\_init** (void \*event)
- **void** **bk\_init32** (void \*event)
- **INT** **bk\_size** (void \*event)
- **void** **bk\_create** (void \*event, const char \*name, WORD type, void \*pdata)
- **INT** **bk\_close** (void \*event, void \*pdata)
- **INT** **bk\_list** (void \*event, char \*bklist)
- **INT** **bk\_locate** (void \*event, const char \*name, void \*pdata)
- **INT** **bk\_find** (**BANK\_HEADER** \*pbkh, const char \*name, DWORD \*bklen, DWORD \*bktype, void \*\*pdata)
- **INT** **bk\_iterate** (void \*event, **BANK** \*\*pbk, void \*pdata)
- **INT** **bk\_swap** (void \*event, BOOL force)
- **INT** **hs\_set\_path** (char \*path)
- **INT** **hs\_open\_file** (DWORD ltime, char \*suffix, 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 \*filename1, char \*buffer1, INT buffer\_size1, char \*filename2, char \*buffer2, INT buffer\_size2, char \*filename3, 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)

## 8.19 midas.dox File Reference

## 8.20 midas.h File Reference

### 8.20.1 Detailed Description

The main include file

Definition in file `midas.h`.

#### Data Structures

- struct **ALARM**
- struct **ALARM\_CLASS**
- struct **ANA\_MODULE**
- struct **ANA\_TEST**
- struct **ANALYZE\_REQUEST**
- struct **AR\_INFO**
- struct **AR\_STATS**
- struct **BANK**
- struct **BANK32**
- struct **BANK\_HEADER**
- struct **BANK\_LIST**
- struct **BUFFER**
- struct **BUFFER\_CLIENT**
- struct **BUFFER\_HEADER**
- struct **BUS\_DRIVER**
- struct **DEF\_RECORD**
- struct **DEVICE\_DRIVER**
- struct **eqpmnt**
- struct **EQUIPMENT\_INFO**
- struct **EQUIPMENT\_STATS**
- struct **EVENT\_HEADER**
- struct **EVENT\_REQUEST**
- struct **HIST\_RECORD**
- struct **HISTORY**
- struct **INDEX\_RECORD**
- struct **KEY**
- struct **KEYLIST**
- struct **PROGRAM\_INFO**
- struct **RUNINFO**
- struct **TAG**

## 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 32`
- `#define MAX_EVENT_REQUESTS 10`
- `#define MAX_OPEN_RECORDS 100`
- `#define MAX_ODB_PATH 256`
- `#define MAX_EXPERIMENT 32`
- `#define BANKLIST_MAX 32`
- `#define STRING_BANKLIST_MAX BANKLIST_MAX * 4`
- `#define STATE_STOPPED 1`
- `#define STATE_PAUSED 2`
- `#define STATE_RUNNING 3`
- `#define FORMAT_MIDAS 1`
- `#define FORMAT_YBOS 2`
- `#define FORMAT_ASCII 3`
- `#define FORMAT_FIXED 4`
- `#define FORMAT_DUMP 5`
- `#define FORMAT_HBOOK 6`
- `#define FORMAT_ROOT 7`
- `#define GET_ALL (1<<0)`
- `#define GET_SOME (1<<1)`
- `#define GET_FARM (1<<2)`
- `#define TID_BYTE 1`
- `#define TID_SBYTE 2`
- `#define TID_CHAR 3`
- `#define TID_WORD 4`
- `#define TID_SHORT 5`
- `#define TID_DWORD 6`
- `#define TID_INT 7`
- `#define TID_BOOL 8`
- `#define TID_FLOAT 9`
- `#define TID_DOUBLE 10`

---

- #define **TID\_BITFIELD** 11
- #define **TID\_STRING** 12
- #define **TID\_ARRAY** 13
- #define **TID\_STRUCT** 14
- #define **TID\_KEY** 15
- #define **TID\_LINK** 16
- #define **TID\_LAST** 17
- #define **SYNC** 0
- #define **MODE\_READ** (1<<0)
- #define **RPC\_OTIMEOUT** 1
- #define **WF\_WATCH\_ME** (1<<0)
- #define **TR\_START** (1<<0)
- #define **TR\_STOP** (1<<1)
- #define **TR\_PAUSE** (1<<2)
- #define **TR\_RESUME** (1<<3)
- #define **EQ\_PERIODIC** (1<<0)
- #define **EQ\_POLLED** (1<<1)
- #define **EQ\_INTERRUPT** (1<<2)
- #define **EQ\_SLOW** (1<<3)
- #define **EQ\_MANUAL\_TRIG** (1<<4)
- #define **EQ\_FRAGMENTED** (1<<5)
- #define **RO\_RUNNING** (1<<0)
- #define **RO\_STOPPED** (1<<1)
- #define **RO\_PAUSED** (1<<2)
- #define **RO\_BOR** (1<<3)
- #define **RO\_EOR** (1<<4)
- #define **RO\_PAUSE** (1<<5)
- #define **RO\_RESUME** (1<<6)
- #define **RO\_TRANSITIONS** (RO\_BOR|RO\_EOR|RO\_PAUSE|RO\_RESUME)
- #define **RO\_ALWAYS** (0xFF)
- #define **RO\_ODB** (1<<8)
- #define **CH\_BS** 8
- #define **LAM\_SOURCE**(c, s) (c<<24 | ((s) & 0xFFFFFFFF))
- #define **LAM\_STATION**(s) (1<<(s-1))
- #define **LAM\_SOURCE\_CRATE**(c) (c>>24)
- #define **LAM\_SOURCE\_STATION**(s) ((s) & 0xFFFFFFFF)
- #define **CNAF** 0x1
- #define **max**(a, b) (((a) > (b)) ? (a) : (b))
- #define **min**(a, b) (((a) < (b)) ? (a) : (b))
- #define **ALIGN**(x) (((x)+7) & ~7)
- #define **VALIGN**(adr, align) (((PTYPE) (adr)+align-1) & ~(align-1))
- #define **MT\_ERROR** (1<<0)

---

- #define **MT\_INFO** (1<<1)
- #define **MT\_DEBUG** (1<<2)
- #define **MT\_USER** (1<<3)
- #define **MT\_LOG** (1<<4)
- #define **MT\_TALK** (1<<5)
- #define **MT\_CALL** (1<<6)
- #define **MT\_ALL** 0xFF
- #define **MERROR** MT\_ERROR, \_\_FILE\_\_, \_\_LINE\_\_
- #define **MINFO** MT\_INFO, \_\_FILE\_\_, \_\_LINE\_\_
- #define **MDEBUG** MT\_DEBUG, \_\_FILE\_\_, \_\_LINE\_\_
- #define **MUSER** MT\_USER, \_\_FILE\_\_, \_\_LINE\_\_
- #define **MLOG** MT\_LOG, \_\_FILE\_\_, \_\_LINE\_\_
- #define **MTALK** MT\_TALK, \_\_FILE\_\_, \_\_LINE\_\_
- #define **MCALL** MT\_CALL, \_\_FILE\_\_, \_\_LINE\_\_
- #define **SUCCESS** 1
- #define **CM\_SUCCESS** 1
- #define **CM\_SET\_ERROR** 102
- #define **CM\_NO\_CLIENT** 103
- #define **CM\_DB\_ERROR** 104
- #define **CM\_UNDEF\_EXP** 105
- #define **CM\_VERSION\_MISMATCH** 106
- #define **CM\_SHUTDOWN** 107
- #define **CM\_WRONG\_PASSWORD** 108
- #define **CM\_UNDEF\_ENVIRON** 109
- #define **CM\_DEFERRED\_TRANSITION** 110
- #define **CM\_TRANSITION\_IN\_PROGRESS** 111
- #define **BM\_SUCCESS** 1
- #define **BM\_CREATED** 202
- #define **BM\_NO\_MEMORY** 203
- #define **BM\_INVALID\_NAME** 204
- #define **BM\_INVALID\_HANDLE** 205
- #define **BM\_NO\_SLOT** 206
- #define **BM\_NO\_MUTEX** 207
- #define **BM\_NOT\_FOUND** 208
- #define **BM\_ASYNC\_RETURN** 209
- #define **BM\_TRUNCATED** 210
- #define **BM\_MULTIPLE\_HOSTS** 211
- #define **BM\_MEMSIZE\_MISMATCH** 212
- #define **BM\_CONFLICT** 213
- #define **BM\_EXIT** 214
- #define **BM\_INVALID\_PARAM** 215
- #define **BM\_MORE\_EVENTS** 216
- #define **BM\_INVALID\_MIXING** 217

- #define **BM\_NO\_SHM** 218
- #define **DB\_SUCCESS** 1
- #define **DB\_CREATED** 302
- #define **DB\_NO\_MEMORY** 303
- #define **DB\_INVALID\_NAME** 304
- #define **DB\_INVALID\_HANDLE** 305
- #define **DB\_NO\_SLOT** 306
- #define **DB\_NO\_MUTEX** 307
- #define **DB\_MEMSIZE\_MISMATCH** 308
- #define **DB\_INVALID\_PARAM** 309
- #define **DB\_FULL** 310
- #define **DB\_KEY\_EXIST** 311
- #define **DB\_NO\_KEY** 312
- #define **DB\_KEY\_CREATED** 313
- #define **DB\_TRUNCATED** 314
- #define **DB\_TYPE\_MISMATCH** 315
- #define **DB\_NO\_MORE\_SUBKEYS** 316
- #define **DB\_FILE\_ERROR** 317
- #define **DB\_NO\_ACCESS** 318
- #define **DB\_STRUCT\_SIZE\_MISMATCH** 319
- #define **DB\_OPEN\_RECORD** 320
- #define **DB\_OUT\_OF\_RANGE** 321
- #define **DB\_INVALID\_LINK** 322
- #define **DB\_CORRUPTED** 323
- #define **DB\_STRUCT\_MISMATCH** 324
- #define **SS\_SUCCESS** 1
- #define **SS\_CREATED** 402
- #define **SS\_NO\_MEMORY** 403
- #define **SS\_INVALID\_NAME** 404
- #define **SS\_INVALID\_HANDLE** 405
- #define **SS\_INVALID\_ADDRESS** 406
- #define **SS\_FILE\_ERROR** 407
- #define **SS\_NO\_MUTEX** 408
- #define **SS\_NO\_PROCESS** 409
- #define **SS\_NO\_THREAD** 410
- #define **SS\_SOCKET\_ERROR** 411
- #define **SS\_TIMEOUT** 412
- #define **SS\_SERVER\_RECV** 413
- #define **SS\_CLIENT\_RECV** 414
- #define **SS\_ABORT** 415
- #define **SS\_EXIT** 416
- #define **SS\_NO\_TAPE** 417
- #define **SS\_DEV\_BUSY** 418



- #define **SS\_IO\_ERROR** 419
- #define **SS\_TAPE\_ERROR** 420
- #define **SS\_NO\_DRIVER** 421
- #define **SS\_END\_OF\_TAPE** 422
- #define **SS\_END\_OF\_FILE** 423
- #define **SS\_FILE\_EXISTS** 424
- #define **SS\_NO\_SPACE** 425
- #define **SS\_INVALID\_FORMAT** 426
- #define **SS\_NO\_ROOT** 427
- #define **RPC\_SUCCESS** 1
- #define **RPC\_ABORT** SS\_ABORT
- #define **RPC\_NO\_CONNECTION** 502
- #define **RPC\_NET\_ERROR** 503
- #define **RPC\_TIMEOUT** 504
- #define **RPC\_EXCEED\_BUFFER** 505
- #define **RPC\_NOT\_REGISTERED** 506
- #define **RPC\_CONNCLOSED** 507
- #define **RPC\_INVALID\_ID** 508
- #define **RPC\_SHUTDOWN** 509
- #define **RPC\_NO\_MEMORY** 510
- #define **RPC\_DOUBLE\_DEFINED** 511
- #define **FE\_SUCCESS** 1
- #define **FE\_ERR\_ODB** 602
- #define **FE\_ERR\_HW** 603
- #define **FE\_ERR\_DISABLED** 604
- #define **FE\_ERR\_DRIVER** 605
- #define **HS\_SUCCESS** 1
- #define **HS\_FILE\_ERROR** 702
- #define **HS\_NO\_MEMORY** 703
- #define **HS\_TRUNCATED** 704
- #define **HS\_WRONG\_INDEX** 705
- #define **HS\_UNDEFINED\_EVENT** 706
- #define **HS\_UNDEFINED\_VAR** 707
- #define **FTP\_SUCCESS** 1
- #define **FTP\_NET\_ERROR** 802
- #define **FTP\_FILE\_ERROR** 803
- #define **FTP\_RESPONSE\_ERROR** 804
- #define **FTP\_INVALID\_ARG** 805
- #define **EL\_SUCCESS** 1
- #define **EL\_FILE\_ERROR** 902
- #define **EL\_NO\_MESSAGE** 903
- #define **EL\_TRUNCATED** 904
- #define **EL\_FIRST\_MSG** 905

- #define **EL\_LAST\_MSG** 906
- #define **AL\_SUCCESS** 1
- #define **AL\_INVALID\_NAME** 1002
- #define **AL\_ERROR\_ODB** 1003
- #define **AL\_RESET** 1004
- #define **CMD\_INIT** (1<<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

## 8.21 mrpc.c File Reference

### 8.21.1 Detailed Description

The Midas RPC file

Definition in file **mrpc.c**.

#### Variables

- RPC\_LIST **rpc\_list\_library** []
- RPC\_LIST **rpc\_list\_system** []

## 8.22 mrpc.h File Reference

### 8.22.1 Detailed Description

The mrpc include file

Definition in file **mrpc.h**.

#### Defines

- #define **RPC\_CM\_SET\_CLIENT\_INFO** 11000
- #define **RPC\_CM\_SET\_WATCHDOG\_PARAMS** 11001
- #define **RPC\_CM\_CLEANUP** 11002
- #define **RPC\_CM\_GET\_WATCHDOG\_INFO** 11003
- #define **RPC\_CM\_MSG\_LOG** 11004
- #define **RPC\_CM\_EXECUTE** 11005
- #define **RPC\_CM\_SYNCHRONIZE** 11006
- #define **RPC\_CM\_ASCTIME** 11007
- #define **RPC\_CM\_TIME** 11008
- #define **RPC\_CM\_MSG** 11009
- #define **RPC\_CM\_EXIST** 11011
- #define **RPC\_CM\_MSG\_RETRIEVE** 11012
- #define **RPC\_CM\_MSG\_LOG1** 11013
- #define **RPC\_BM\_OPEN\_BUFFER** 11100
- #define **RPC\_BM\_CLOSE\_BUFFER** 11101
- #define **RPC\_BM\_CLOSE\_ALL\_BUFFERS** 11102
- #define **RPC\_BM\_GET\_BUFFER\_INFO** 11103
- #define **RPC\_BM\_GET\_BUFFER\_LEVEL** 11104
- #define **RPC\_BM\_INIT\_BUFFER\_COUNTERS** 11105
- #define **RPC\_BM\_SET\_CACHE\_SIZE** 11106
- #define **RPC\_BM\_ADD\_EVENT\_REQUEST** 11107
- #define **RPC\_BM\_REMOVE\_EVENT\_REQUEST** 11108
- #define **RPC\_BM\_SEND\_EVENT** 11109
- #define **RPC\_BM\_FLUSH\_CACHE** 11110
- #define **RPC\_BM\_RECEIVE\_EVENT** 11111
- #define **RPC\_BM\_MARK\_READ\_WAITING** 11112
- #define **RPC\_BM\_EMPTY\_BUFFERS** 11113
- #define **RPC\_BM\_SKIP\_EVENT** 11114
- #define **RPC\_DB\_OPEN\_DATABASE** 11200
- #define **RPC\_DB\_CLOSE\_DATABASE** 11201
- #define **RPC\_DB\_CLOSE\_ALL\_DATABASES** 11202
- #define **RPC\_DB\_CREATE\_KEY** 11203
- #define **RPC\_DB\_CREATE\_LINK** 11204

- #define **RPC\_DB\_SET\_VALUE** 11205
- #define **RPC\_DB\_GET\_VALUE** 11206
- #define **RPC\_DB\_FIND\_KEY** 11207
- #define **RPC\_DB\_FIND\_LINK** 11208
- #define **RPC\_DB\_GET\_PATH** 11209
- #define **RPC\_DB\_DELETE\_KEY** 11210
- #define **RPC\_DB\_ENUM\_KEY** 11211
- #define **RPC\_DB\_GET\_KEY** 11212
- #define **RPC\_DB\_GET\_DATA** 11213
- #define **RPC\_DB\_SET\_DATA** 11214
- #define **RPC\_DB\_SET\_DATA\_INDEX** 11215
- #define **RPC\_DB\_SET\_MODE** 11216
- #define **RPC\_DB\_GET\_RECORD\_SIZE** 11219
- #define **RPC\_DB\_GET\_RECORD** 11220
- #define **RPC\_DB\_SET\_RECORD** 11221
- #define **RPC\_DB\_ADD\_OPEN\_RECORD** 11222
- #define **RPC\_DB\_REMOVE\_OPEN\_RECORD** 11223
- #define **RPC\_DB\_SAVE** 11224
- #define **RPC\_DB\_LOAD** 11225
- #define **RPC\_DB\_SET\_CLIENT\_NAME** 11226
- #define **RPC\_DB\_RENAME\_KEY** 11227
- #define **RPC\_DB\_ENUM\_LINK** 11228
- #define **RPC\_DB\_REORDER\_KEY** 11229
- #define **RPC\_DB\_CREATE\_RECORD** 11230
- #define **RPC\_DB\_GET\_DATA\_INDEX** 11231
- #define **RPC\_DB\_GET\_KEY\_TIME** 11232
- #define **RPC\_DB\_GET\_OPEN\_RECORDS** 11233
- #define **RPC\_DB\_FLUSH\_DATABASE** 11235
- #define **RPC\_DB\_SET\_DATA\_INDEX2** 11236
- #define **RPC\_DB\_GET\_KEY\_INFO** 11237
- #define **RPC\_DB\_GET\_DATA1** 11238
- #define **RPC\_DB\_SET\_NUM\_VALUES** 11239
- #define **RPC\_DB\_CHECK\_RECORD** 11240
- #define **RPC\_DB\_GET\_NEXT\_LINK** 11241
- #define **RPC\_HS\_SET\_PATH** 11300
- #define **RPC\_HS\_DEFINE\_EVENT** 11301
- #define **RPC\_HS\_WRITE\_EVENT** 11302
- #define **RPC\_HS\_COUNT\_EVENTS** 11303
- #define **RPC\_HS\_ENUM\_EVENTS** 11304
- #define **RPC\_HS\_COUNT\_VARS** 11305
- #define **RPC\_HS\_ENUM\_VARS** 11306
- #define **RPC\_HS\_READ** 11307
- #define **RPC\_HS\_GET\_VAR** 11308

- #define **RPC\_HS\_GET\_EVENT\_ID** 11309
- #define **RPC\_EL\_SUBMIT** 11400
- #define **RPC\_AL\_CHECK** 11500
- #define **RPC\_AL\_TRIGGER\_ALARM** 11501
- #define **RPC\_RC\_TRANSITION** 12000
- #define **RPC\_ANA\_CLEAR\_HISTOS** 13000
- #define **RPC\_LOG\_REWIND** 14000
- #define **RPC\_TEST** 15000
- #define **RPC\_CNAF16** 16000
- #define **RPC\_CNAF24** 16001
- #define **RPC\_MANUAL\_TRIG** 17000
- #define **RPC\_ID\_WATCHDOG** 99997
- #define **RPC\_ID\_SHUTDOWN** 99998
- #define **RPC\_ID\_EXIT** 99999

## 8.23 msystem.h File Reference

### Data Structures

- struct **DATABASE**
- struct **DATABASE\_CLIENT**
- struct **DATABASE\_HEADER**
- struct **FREE\_DESCRIP**
- struct **OPEN\_RECORD**
- struct **RECORD\_LIST**
- struct **REQUEST\_LIST**

### Defines

- #define **DRI\_16** (1<<0)
- #define **DRI\_32** (1<<1)
- #define **DRI\_64** (1<<2)
- #define **DRI\_LITTLE\_ENDIAN** (1<<3)
- #define **DRI\_BIG\_ENDIAN** (1<<4)
- #define **DRF\_IEEE** (1<<5)
- #define **DRF\_G\_FLOAT** (1<<6)
- #define **DR\_ASCII** (1<<7)
- #define **WORD\_SWAP**(x)
- #define **DWORD\_SWAP**(x)
- #define **QWORD\_SWAP**(x)

## 8.24 mvmestd.h File Reference

### 8.24.1 Detailed Description

The Midas VME include file

Definition in file **mvmestd.h**.

#### Defines

- `#define VME_AMOD_A32_SB (0x0F)`
- `#define VME_AMOD_A32_SP (0x0E)`
- `#define VME_AMOD_A32_SD (0x0D)`
- `#define VME_AMOD_A32_NB (0x0B)`
- `#define VME_AMOD_A32_NP (0x0A)`
- `#define VME_AMOD_A32_ND (0x09)`
- `#define VME_AMOD_A32_SMBLT (0x0C)`
- `#define VME_AMOD_A32_NMBLT (0x08)`
- `#define VME_AMOD_A24_SB (0x3F)`
- `#define VME_AMOD_A24_SP (0x3E)`
- `#define VME_AMOD_A24_SD (0x3D)`
- `#define VME_AMOD_A24_NB (0x3B)`
- `#define VME_AMOD_A24_NP (0x3A)`
- `#define VME_AMOD_A24_ND (0x39)`
- `#define VME_AMOD_A24_SMBLT (0x3C)`
- `#define VME_AMOD_A24_NMBLT (0x38)`
- `#define VME_AMOD_A16_SD (0x2D)`
- `#define VME_AMOD_A16_ND (0x29)`

#### Functions

- `int EXPRT vme_open (int device, int mode)`
- `int EXPRT vme_ioctl (int vh, int req, int *parm)`
- `int EXPRT vme_close (int vh)`
- `int EXPRT vme_read (int vh, void *dst, int vme_addr, int am, int size, int dma)`
- `int EXPRT vme_write (int vh, void *src, int vme_addr, int am, int size, int dma)`
- `int EXPRT vme_mmap (int vh, void **ptr, int vme_addr, int size)`
- `int EXPRT vme_unmap (int vh, void *ptr, int size)`



## 8.25 newdocfeatures.dox File Reference

## 8.26 odb.c File Reference

### 8.26.1 Detailed Description

The Online Database file

Definition in file **odb.c**.

#### Functions

- **INT db\_open\_database** (char \*database\_name, INT database\_size, HANDLE \*hDB, char \*client\_name)
- **INT db\_close\_database** (HANDLE hDB)
- **INT db\_lock\_database** (HANDLE hDB)
- **INT db\_unlock\_database** (HANDLE hDB)
- **INT db\_protect\_database** (HANDLE hDB)
- **INT db\_create\_key** (HANDLE hDB, HANDLE hKey, char \*key\_name, DWORD type)
- **INT db\_create\_link** (HANDLE hDB, HANDLE hKey, char \*link\_name, char \*destination)
- **INT db\_delete\_key1** (HANDLE hDB, HANDLE hKey, INT level, BOOL follow\_links)
- **INT db\_delete\_key** (HANDLE hDB, HANDLE hKey, BOOL follow\_links)
- **INT db\_find\_key** (HANDLE hDB, HANDLE hKey, char \*key\_name, HANDLE \*subhKey)
- **INT db\_set\_value** (HANDLE hDB, HANDLE hKeyRoot, char \*key\_name, void \*data, INT data\_size, INT num\_values, DWORD type)
- **INT db\_get\_value** (HANDLE hDB, HANDLE hKeyRoot, char \*key\_name, void \*data, INT \*buf\_size, DWORD type, BOOL create)
- **INT db\_enum\_key** (HANDLE hDB, HANDLE hKey, INT index, HANDLE \*subkey\_handle)
- **INT db\_get\_key** (HANDLE hDB, HANDLE hKey, **KEY** \*key)
- **INT db\_get\_key\_time** (HANDLE hDB, HANDLE hKey, DWORD \*delta)
- **INT db\_get\_key\_info** (HANDLE hDB, HANDLE hKey, char \*name, INT name\_size, INT \*type, INT \*num\_values, INT \*item\_size)
- **INT db\_get\_data** (HANDLE hDB, HANDLE hKey, void \*data, INT \*buf\_size, DWORD type)
- **INT db\_get\_data\_index** (HANDLE hDB, HANDLE hKey, void \*data, INT \*buf\_size, INT index, DWORD type)
- **INT db\_set\_data** (HANDLE hDB, HANDLE hKey, void \*data, INT buf\_size, INT num\_values, DWORD type)

- **INT db\_set\_data\_index** (HANDLE hDB, HANDLE hKey, void \*data, INT data\_size, INT index, DWORD type)
- **INT db\_load** (HANDLE hDB, HANDLE hKeyRoot, char \*filename, BOOL bRemote)
- **INT db\_copy** (HANDLE hDB, HANDLE hKey, char \*buffer, INT \*buffer\_size, char \*path)
- **INT db\_paste** (HANDLE hDB, HANDLE hKeyRoot, char \*buffer)
- **INT db\_save** (HANDLE hDB, HANDLE hKey, char \*filename, BOOL bRemote)
- **INT db\_save\_struct** (HANDLE hDB, HANDLE 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** (HANDLE hDB, HANDLE hKey, INT align, INT \*buf\_size)
- **INT db\_get\_record** (HANDLE hDB, HANDLE hKey, void \*data, INT \*buf\_size, INT align)
- **INT db\_set\_record** (HANDLE hDB, HANDLE hKey, void \*data, INT buf\_size, INT align)
- **INT db\_create\_record** (HANDLE hDB, HANDLE hKey, char \*orig\_key\_name, char \*init\_str)
- **INT db\_check\_record** (HANDLE hDB, HANDLE hKey, char \*keyname, char \*rec\_str, BOOL correct)
- **INT db\_open\_record** (HANDLE hDB, HANDLE hKey, void \*ptr, INT rec\_size, WORD access\_mode, void(\*dispatcher)(INT, INT, void \*), void \*info)
- **INT db\_close\_record** (HANDLE hDB, HANDLE hKey)
- **INT db\_close\_all\_records** ()
- **INT db\_update\_record** (INT hDB, INT hKey, int socket)
- **INT db\_send\_changed\_records** ()

## 8.27 odbstruct.dox File Reference

---

## 8.28 quickstart.dox File Reference

## 8.29 system.c File Reference

### 8.29.1 Detailed Description

The Midas System file

Definition in file **system.c**.

#### Functions

- **INT ss\_thread\_create** (INT(\*thread\_func)(void \*), void \*param)
- **INT ss\_thread\_kill** (INT thread\_id)
- **DWORD ss\_millitime** ()
- **DWORD ss\_time** ()
- **INT ss\_sleep** (INT millisec)

## 8.30 utilities.dox File Reference

## 8.31 ybos.c File Reference

### 8.31.1 Detailed Description

The YBOS file

Definition in file **ybos.c**.

#### Functions

- void **ybk\_init** (DWORD \*plrl)
- void **ybk\_create** (DWORD \*plrl, char \*bkname, DWORD bktype, void \*pbkdat)
- INT **ybk\_close** (DWORD \*plrl, void \*pbkdat)
- INT **ybk\_size** (DWORD \*plrl)
- INT **ybk\_list** (DWORD \*plrl, char \*bklist)
- INT **ybk\_find** (DWORD \*plrl, char \*bkname, DWORD \*bklen, DWORD \*bktype, void \*\*pbk)
- INT **ybk\_locate** (DWORD \*plrl, char \*bkname, void \*pdata)
- INT **ybk\_iterate** (DWORD \*plrl, YBOS\_BANK\_HEADER \*\*pybkh, void \*\*pdata)



## 8.32 ybos.h File Reference

### 8.32.1 Detailed Description

The YBOS include file

Definition in file **ybos.h**.

#### Defines

- #define **YBOS\_PHYREC\_SIZE** 8192
- #define **YBOS\_BUFFER\_SIZE** 3\*(YBOS\_PHYREC\_SIZE<<2) + MAX\_EVENT\_SIZE + 128
- #define **YB\_BANKLIST\_MAX** 32
- #define **YB\_STRING\_BANKLIST\_MAX** YB\_BANKLIST\_MAX \* 4
- #define **YB\_SUCCESS** 1
- #define **YB\_EVENT\_NOT\_SWAPPED** 2
- #define **YB\_DONE** 2
- #define **YB\_WRONG\_BANK\_TYPE** -100
- #define **YB\_BANK\_NOT\_FOUND** -101
- #define **YB\_SWAP\_ERROR** -102
- #define **YB\_NOMORE\_SLOT** -103
- #define **YB\_UNKNOWN\_FORMAT** -104
- #define **H\_BLOCK\_SIZE** 0
- #define **H\_BLOCK\_NUM** 1
- #define **H\_HEAD\_LEN** 2
- #define **H\_START** 3
- #define **D\_RECORD** 1
- #define **D\_HEADER** 2
- #define **D\_EVTLEN** 3
- #define **YB\_COMPLETE** 1
- #define **YB\_INCOMPLETE** 2
- #define **YB\_NO\_RECOVER** -1
- #define **YB\_NO\_RUN** 0
- #define **YB\_ADD\_RUN** 1
- #define **DSP\_RAW** 1
- #define **DSP\_BANK** 2
- #define **DSP\_UNK** 0
- #define **DSP\_DEC** 1
- #define **DSP\_HEX** 2
- #define **DSP\_ASC** 3
- #define **SWAP\_D2WORD**(\_d2w)

- `#define EVID_TRINAT`
- `#define YBOS_EVID_BANK(__a, __b, __c, __d, __e)`
- `#define MIDAS_EVID_BANK(__a, __b, __c, __d, __e)`
- `#define I2_BKTYPE 1`
- `#define A1_BKTYPE 2`
- `#define I4_BKTYPE 3`
- `#define F4_BKTYPE 4`
- `#define D8_BKTYPE 5`
- `#define I1_BKTYPE 8`
- `#define MAX_BKTYPE I1_BKTYPE+1`

## Chapter 9

# Midas Page Documentation

### 9.1 MIDAS Analyzer

Users can write their own analyzer from scratch or use the standard MIDAS analyzer framework which uses the HBOOK package for histogramming. Using the MIDAS analyzer framework has following advantages: Events are received automatically, only a user routine has to be written to process the events. This concept is similar to the frontend. The analyzer is structured into "stages", where each stage analyzes a part of the event and adds some calculated data to it, which can be read by later stages. This simplifies the design of complex analyzers. The analyzer framework can receive events from a MIDAS buffer (online analysis) or from a file (off-line-analysis) without recompilation. The analyzer framework can produce output files which may contain a combination of raw and analyzed data. Output files can be in different formats like HBOOK RZ files which can be directly analyzed with PAW or .root for ROOT access. An ODB dump contained in a data file can be retrieved and copied to the current ODB. This ensures that the same configuration values are used online and off-line. Additionally, parameters can be overloaded from off-line configuration files. Several files can be analyzed off-line each having its own configuration file. While HBOOK histograms have to be booked and filled manually from the user code, N-tuples can be booked automatically from one or more banks. This works also online where "live" N-tuples can be used to monitor an experiment with PAW. The following paragraphs explain these features in more detail and show how to use them.

#### 9.1.1 The ROOT Midas Analyzer

To be written

### 9.1.2 The PAW/HBOOK Midas Analyzer

To be written

## 9.2 Data format

Utilities(p.??) - Top(p.??) - Supported hardware(p.??)

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

- Midas format(p.??)

- YBOS format(p.??)

### 9.2.1 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()**(p.??) routine and used by the consumers to distinguish between different events. The header is defined in the **EVENT\_HEADER**(p.??) structure in **midas.h**(p.??). It has the following structure:

Event and bank headers with data block.

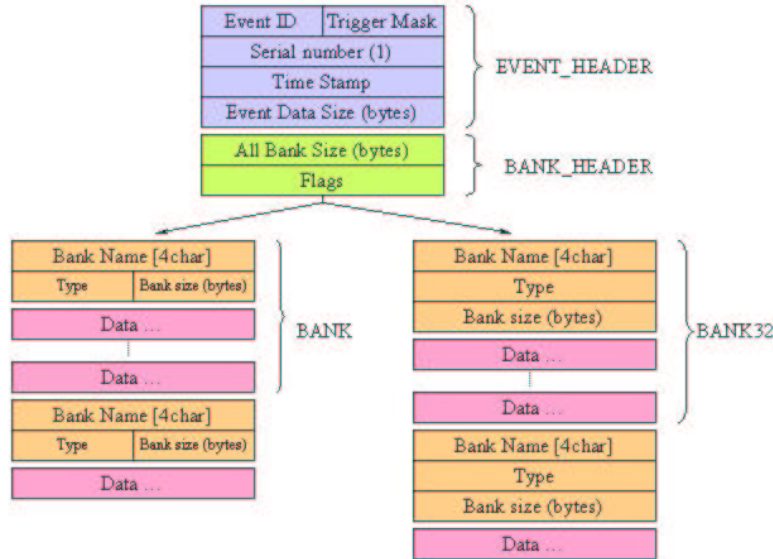


Figure 9.1: Event and bank headers with data block.

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 specific 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 identification, a bank type that is one of the TID\_XXX values defined in `midas.h`(p.??), 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`(p.??) can be used.

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

### 9.2.2 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 defines specific ways for:

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

YBOS is organized on a 4-byte alignment structure.

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

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

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

The YBOS implementation under Midas has the following restrictions:

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

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

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

Ybos Event and bank headers with data block.





Figure 9.2: Ybos Event and bank headers with data block.

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

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

Ybos Physical record structure with data block.

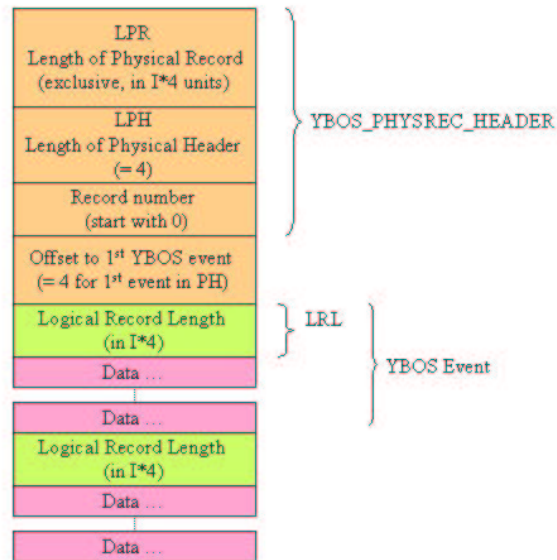


Figure 9.3: Ybos Physical record structure with data block..

The Offset is computed with the following rules:

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

**Utilities(p. ??) - Top(p. ??) - Supported hardware(p. ??)**

## 9.3 Supported hardware

**Data format(p.??) - Top(p.??) - CAMAC and VME access function call(p.??)**

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(p.??)** 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
<b>generic.c</b>	<b>nulldev.c</b>	<b>null.c</b>	<b>caenv488.c</b>
<b>hv.c</b>	<b>epics_ca.c</b>	<b>vxVME.c</b>	<b>lrs1151.c</b>
<b>multic</b>	<b>lrs1454.c</b>	<b>camacrpc.c, camacnul.c</b>	<b>lrs1190.c</b>
<b>slowdev.c</b>	<b>dastemp.c</b>	<b>hyt1331.c</b>	<b>lrs2365.c</b>
	<b>lrs1440.c</b>	<b>kcs2926.c kcs2927.c</b>	<b>lrs2373.c</b>
	<b>bb_psi.c</b>	<b>jorway73a.c</b>	<b>ps7106.c</b>
	<b>lrs4032.c</b>	<b>wecc32.c</b>	<b>sis3803.c</b>
	<b>lcwp950.c</b>	<b>camaclx.c</b>	<b>sis3700.c</b>
	<b>mscbdev.c</b>	<b>dsp004.c</b>	<b>vmeio.c</b>
	<b>lrs2415.c</b>	<b>ces8210.c</b>	
	<b>nitronic.c</b>	<b>rs232.c</b>	
	<b>caen170a.c</b>	<b>tcpip.c</b>	
	<b>das1600.c</b>	<b>cc7700pci.c</b>	
		<b>esone.c</b>	
		<b>ces2117.c</b>	
		<b>lrs1821.c</b>	
		<b>str340.c</b>	
		<b>bt617.c</b>	

Figure 9.4: Class, Device, Bus and Divers Driver listing

- **CAMAC drivers(p.??)**
- **VME drivers(p.??)**

- **GPIB drivers**(p. ??)
- **Other drivers**(p. ??)

### 9.3.1 CAMAC drivers

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

- **CAMAC controllers**
  - [hyt1331.c] This interface uses an ISA board to connect to the crate controller. This card implement a "fast" readout cycle by re-triggering the CAMAC read at the end of the previous one. This feature is unfortunately not reliable when fast processor is used. Wrong returned data can be expected when CPU clocks is above 250MHz. Attempt on "slowing down" the IO through software has not guaranteed perfect result. Contact has been taken with HYTEC in order to see if possible fix can be applied to the interface. First revision of the PC-card PAL has been tested but did not show improvement. CVS version of the hyt1331.c until 1.2 contains "fast readout cycle" and should not be trusted. CVS 1.3 driver revision contains a patch to this problem. In the mean time you can apply your own patch (see **Frequently Asked Questions**(p. ??)) and also Hytec)
  - [hyt1331.c Version >= 1.8.3] This version has been modified for 5331 PCI card support running under the **dio task**(p. ??).
  - [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 Red-Hat 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 *camacx.c*. This requires to first load a module *camac-kcs292x.o*. This software is available but not part of the Midas distribution yet. Please contact [midas@triumf.ca](mailto: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. WindowsNT 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(p.??)** and **mcstd.h(p.??)**, 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(p.??)** 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(p.??)**, **mhttpd task(p.??)** utilities.

### 9.3.2 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(p.??)** files.

- PCI/VME interface

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

**vxVME.c** mvmestd implementation for VxWorks Operating System. Does require cross compiler for the VxWorks target hardware processor and proper WindRiver license.

- VME modules

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

### 9.3.3 GPIB drivers

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

For GPIB Linux support please refer to **The Linux Lab Project**

### 9.3.4 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(p.??)**).

**Data format(p.??) - Top(p.??) - CAMAC and VME access function call(p.??)**

## 9.4 CAMAC and VME access function call

### Supported hardware(p.??) - Top(p.??) - Midas build options and operation considerations(p.??)

Midas defines 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 file **mcstd.h**(p.??) (Midas Camac Standard) provides the interface for the CAMAC access, the file **mvmestd.h**(p.??) (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**(p.??)).

### 9.4.1 Midas CAMAC standard functions

Please refer to the file below for function description. **mcstd.h**(p.??)

### 9.4.2 ESONE CAMAC standard functions

**Not all the functionality of ESONE standard have been fully tested**

Please refer to the file for function description.

**esone.c**(p.??)

### 9.4.3 Midas VME standard functions

This interface is under revision for providing basic VME access through a independent software interface. Please refer to the file below for specific function explanation.

**mvmestd.h**(p.??)

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

Latched trigger layout.

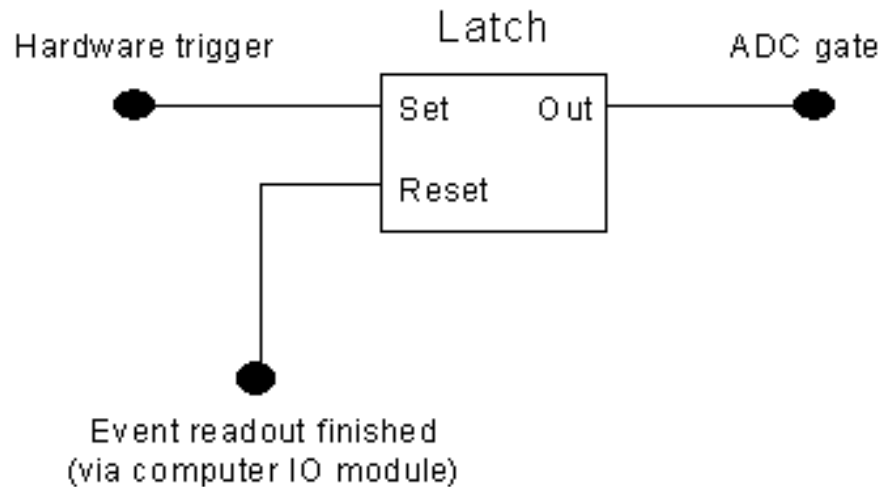


Figure 9.5: Latched trigger layout.

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



Improved Latched trigger layout.

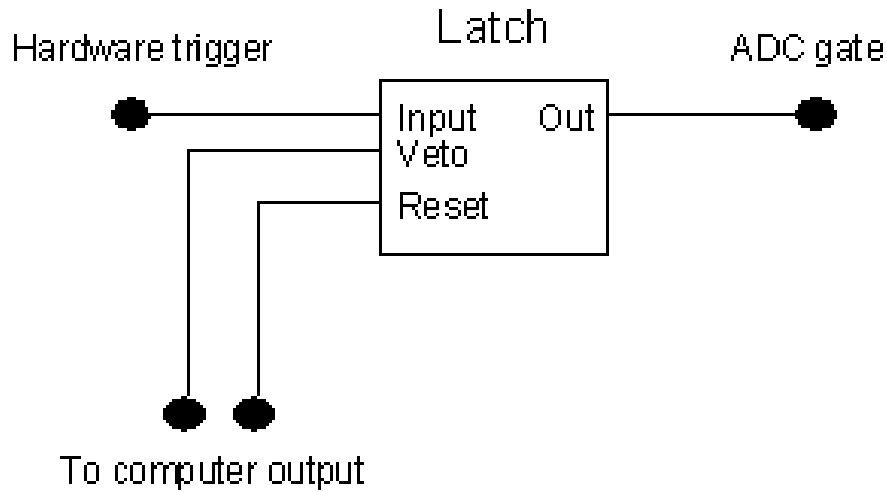


Figure 9.6: Improved Latched trigger layout.

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.

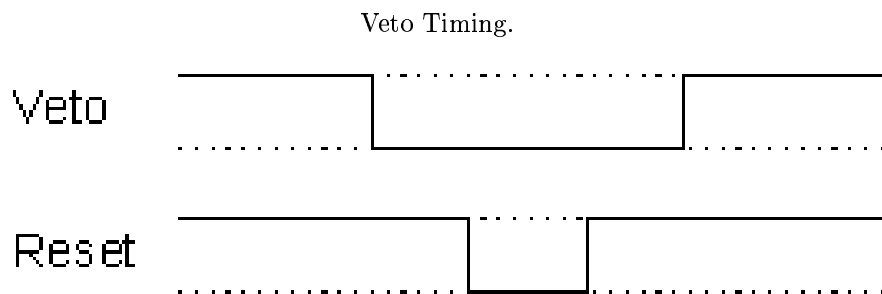


Figure 9.7: Veto Timing.

**Supported hardware(p.??) - Top(p.??) - Midas build options and operation considerations(p.??)**

## 9.5 Midas build options and operation considerations

### CAMAC and VME access function call(p.??) - Top(p.??) - Midas Code and Libraries(p.??)

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

### 9.5.1 Building Options

By default Midas is build with a minimum of pre-compiler flags. But the Makefile contains options for the user to apply customization by enabling internal options already available in the package.

- **YBOS\_VERSION\_3\_3**(p.??) , **EVID\_TWIST** , **INCLUDE\_FTPLIB**(p.??) , **INCLUDE\_ZLIB**(p.??) , **SPECIFIC\_OS\_PRG**(p.??)

Other flags are available at the application level:

- **HAVE\_CAMAC**(p.??) , **HAVE\_ROOT**(p.??) , **HAVE\_HBOOK**(p.??) , **USE\_EVENT\_CHANNEL**(p.??) , **DM\_DUAL\_THREAD**(p.??) , **USE\_INT**(p.??)

By default the midas applications are built for use with dynamic library **lib-midas.so**. If static build is required the whole package can be built using the option **static**

```
make static
```

For development it could be useful to build individual application in static. This can be done using the **USERFLAGS**(p.??) option such as:

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

The current OS support is done through fix flag established in the general Makefile. Currently the OS supported are:

- **OS\_OSF1** , **OS\_ULTRIX** , **OS\_FREEBSD** , **OS\_LINUX** , **OS\_SOLARIS**.

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

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

Other OS supported are:

- **OS\_WINNT** : See file `makefile.nt`.
- **OS\_VXWORKS** : See file `makefile.ppc_tri`.

### 9.5.2 USERFLAGS

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

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

### 9.5.3 MIDAS\_PREF\_FLAGS

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

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

### 9.5.4 HAVE\_CAMAC

This flag enable the CAMAC RPC service within the frontend code. The application **mcnaf task(p.??)** and the web **CNAF page(p.??)** are then enabled (default undefined).

### 9.5.5 HAVE\_ROOT

This flag is used for the midas **analyzer task(p.??)** in the case **ROOT** environment is required. An example of the makefile resides in **examples/experiment/Makefile**. This flag is enabled by the presence of a valid **ROOTSYS** environment variable. In the case **ROOTSYS** is not found the analyzer is build without **ROOT** support. In this later case, the application **rmidas task(p.??)** will be missing. (See also **MANA\_LITE(p.??)**).

### 9.5.6 HAVE\_HBOOK

This flag is used for **examples/hbook/Makefile** for building the midas **analyzer task(p.??)** against HBOOK and PAW. In the case the HBOOK library is not found the analyzer will be build without HBOOK support.

In this later case, this "lite" version of the analyzer without CERN library or ROOT, no histograms nor Ntuples will be generated. This allow the user to implement his/her own graphical interface. This "lite" version provides a "analyzer like" environment for monitoring tasks where results may be put back in the ODB or pushed to a secondary logging channel. This flag affect: **mana.c**, **fal.c**, **analyzer.c**, **adccalib.c**, **adcsum.c** from the default distribution.

### 9.5.7 SPECIFIC\_OS\_PRG

This flag is for internal Makefile preference. Used in particular for additional applications build based on the OS selection. In the example below **mspeaker**, **mlxspeaker task(p.??)** and **dio task(p.??)** are built only under OS\_LINUX.

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

### 9.5.8 INCLUDE\_FTPLIB

FTP support "INCLUDE\_FTPLIB" Application such as the **mlogger task(p.??)**, **lazylogger task(p.??)** can use the ftp channel for data transfer.

### 9.5.9 INCLUDE\_ZLIB

The applications **lazylogger task(p.??)**, **mdump task(p.??)** can be built with **zlib.a** in order to gain direct access to the data files with the extension **mid.gz** or **ybs.gz** (module **ybos.c(p.??)**). Building the application static prevent to rebuild the whole midas package. The analyzer **mana** has this option already enabled.

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

### 9.5.10 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(p.??)**

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

### 9.5.11 DM\_DUAL\_THREAD

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

### 9.5.12 USE\_EVENT\_CHANNEL

To be used in conjunction with the `DM_DUAL_THREAD`(p.??).

### 9.5.13 USE\_INT

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

### 9.5.14 Environment variables

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

#### 9.5.14.1 MIDAS\_EXPTAB

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

#### 9.5.14.2 MIDAS\_SERVER\_HOST

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

#### 9.5.14.3 MIDAS\_EXPT\_NAME

This variable predefines the name of the experiment to connect by default. It prevents the requested application to ask for the experiment name when multiple experiments are available on the host or to add the `-e <expt_name>` argument

to the application command. This variable is valid for Unix as well as Windows OS.

#### 9.5.14.4 MIDAS\_DIR

This variable predefines the LOCAL directory path where the shared memories for the experiment are located. It supersedes the host\_name and the expt\_name as well as the MIDAS\_SERVER\_HOST(p.??) and MIDAS\_EXPT\_NAME(p.??) as a given directory path can only refer to a single experiment.

#### 9.5.14.5 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(p.??) - Top(p.??) - Midas Code and Libraries(p.??)**

## 9.6 Midas Code and Libraries

**Midas build options and operation considerations**(p. ??) - **Top**(p. ??) - **Frequently Asked Questions**(p. ??)

This section covers several aspect of the Midas system.

- **State Codes & Transition Codes**(p. ??)
- **Midas Data Types**(p. ??)
  - **Midas bank examples**(p. ??)
- **YBOS Bank Types**(p. ??)
  - **YBOS bank examples**(p. ??)
- **Midas Code and Libraries**(p. ??)

### 9.6.1 State Codes & Transition Codes

- **State Codes** : These number will be apparent in the ODB under the **ODB /RunInfo Tree**(p. ??).
  - STATE\_STOPPED
  - STATE\_PAUSED
  - STATE\_RUNNING
- **Transition Codes** These number will be apparent in the ODB under the **ODB /RunInfo Tree**(p. ??).
  - TR\_START
  - TR\_STOP
  - TR\_PAUSE
  - TR\_RESUME

### 9.6.2 Midas Data Types

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

- **BYTE** unsigned char
- **WORD** unsigned short int (16bits word)

- DWORD unsigned 32bits word
- INT signed 32bits word
- BOOL OS dependent.

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

- TID\_BYTE unsigned byte 0 255
- TID\_SBYTE signed BYTE -128 127
- TID\_CHAR single character 0 255
- TID\_WORD two BYTE 0 65535
- TID\_SHORT signed WORD -32768 32767
- TID\_DWORD four bytes 0  $2^{32}-1$
- TID\_INT signed DWORD  $-2^{31}$   $2^{31}-1$
- TID\_BOOL four bytes bool 0 1
- TID\_FLOAT four bytes float format
- TID\_DOUBLE eight bytes float format

### 9.6.3 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...

### 9.6.4 YBOS Bank Types

YBOS defines several type but all types should be 4 bytes aligned. Distinction of signed and unsigned is not done. When mixing MIDAS and YBOS in the frontend for RO\_ODB see **The Equipment structure**(p.??) make sure the bank types are compatible (see also **YBOS** `#define(p.??)`)

- **I1\_BKTYPE** Bank of Bytes



- **I2\_BKTYPE** Bank of 2 bytes data
- **I4\_BKTYPE** Bank of 4 bytes data
- **F4\_BKTYPE** Bank of float data
- **D8\_BKTYPE** Bank of double data
- **A1\_BKTYPE** Bank of ASCII char

### 9.6.5 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 containing a duplicate of the midas header. As the Midas header is stripped out of the event when data are logger, it is necessary to compose such event to retain event information for off-line

analysis. Uses of predefined macros (see **Midas Code and Libraries**(p.??)) are available in order to extract from a pre-composed Midas event the internal header fields i.e. Event ID, Trigger mask, Serial number, Time stamp. In this EVID bank we added the current run number which is retrieve by the frontend at the begin of a run.

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

```
INT read_cum_scaler_event(char *pevent, INT off)
{
    INT i;
    DWORD *pbkdat, *pbktop, *podbvar;

    ybk_init((DWORD *) pevent);

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

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

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

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

- **Backend code** If the data logging is done through YBOS format (see **ODB /Logger Tree**(p.??) Format) the events on the storage media will have been stripped from the MIDAS header used for transferring 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 *plr1, *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);
    }
    ...
    ...
}

```

### 9.6.6 Midas Code and Libraries

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

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

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

- **al\_**xxx(...) : Alarm system calls
- **bk\_**xxx(...) : Midas bank manipulation calls
- **bm\_**xxx(...) : Buffer management calls
- **cm\_**xxx(...) : Common system calls
- **db\_**xxx(...) : Database management calls
- **el\_**xxx(...) : Electronic Log book calls
- **hs\_**xxx(...) : History manipulation calls
- **ss\_**xxx(...) : System calls
- **ybk\_**xxx(...) : YBOS bank manipulation

### 9.6.7 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**(p.??). All of them are defined in the **Midas Macros**(p.??), **System Macros**(p.??), **YBOS Macros**(p.??) header files.

- **Message Macros.** These Macros compact the 3 first arguments of the **cm\_msg()**(p.??) call. It replaces the type of message, the routine name and the line number in the C-code. See example in **cm\_msg()**(p.??).
  - **MERROR**(p.??) : For error (MT\_ERROR, \_\_FILE\_\_, \_\_LINE\_\_)
  - **MINFO**(p.??) : For info (MT\_INFO, \_\_FILE\_\_, \_\_LINE\_\_)
  - **MDEBUG**(p.??) : For debug (MT\_DEBUG, \_\_FILE\_\_, \_\_LINE\_\_)
  - **MUSER**(p.??) : Produced by interactive user (MT\_USER, \_\_FILE\_\_, \_\_LINE\_\_)
  - **MLOG**(p.??) : Info message which is only logged (MT\_LOG, \_\_FILE\_\_, \_\_LINE\_\_)
  - **MTALK**(p.??) : Info message for speech system (MT\_TALK, \_\_FILE\_\_, \_\_LINE\_\_)
  - **MCALL**(p.??) : 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. Usually the CAMAC Crate Controller 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 masked LAM register).

The `poll_event()(p.??)` uses this feature and return a variable which contains a bit-wise value of the current LAM register in the Crate Controller.

- `LAM_SOURCE(p.??)`
- `LAM_STATION(p.??)`
- `LAM_SOURCE_CRATE(p.??)`
- `LAM_SOURCE_STATION(p.??)`
- **BYTE swap manipulation** These Macros can be used in the backend analyzer when **little-endian**/**big-endian** are mixed in the event.
  - `WORD_SWAP(p.??)`
  - `DWORD_SWAP(p.??)`
  - `QWORD_SWAP(p.??)`

**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 requirement. 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(p.??)`
- `EVENT_ID(p.??)`
- `SERIAL_NUMBER(p.??)`
- `TIME_STAMP(p.??)`
  - 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--;  
    ...  
}  
...
```

#### 9.6.7.1 YBOS library

Exportable ybos functions through inclusion of **ybos.h**(p.??)

**Midas build options and operation considerations**(p.??) - **Top**(p.??) -  
**Frequently Asked Questions**(p.??)

## 9.7 Frequently Asked Questions

**Midas Code and Libraries**(p. ??) - **Top**(p. ??) - **Data format**(p. ??)

Feel free to ask questions to one of us ( Stefan Ritt , Pierre-Andr e Audaudruz) or visit the Midas Forum

### 1. Why the CAMAC frontend generate a core dump (linux)?

- If you're not using a Linux driver for the CAMAC access, you need to start the CAMAC frontend application through the task launcher first. See **dio task**(p. ??) or **mcnaf task**(p. ??). This task launcher will grant you access permission to the IO port mapped to your CAMAC interface.

### 2. Where does Midas log file resides?

- As soon as any midas application is started, a file midas.log is produced. The location of this file depends on the setup of the experiment.
- (a) if **exptab** is present and contains the experiment name with the corresponding directory, this is where the file **midas.log** will reside.
- (b) if the midas logger **mlogger task**(p. ??) is running the midas.log will be in the directory pointed by the "Data Dir" key under the /logger key in the ODB tree.
- (c) Otherwise the file midas.log will be created in the current directory in which the Midas application is started.

### 3. How do I protect my experiment from being controlled by aliases?

- Every experiment may have a dedicated password for accessing the experiment from the web browser. This is setup through the ODBedit program with the command **webpass**. This will create a **Security** tree under **/Experiment** with a new key **Web Password** with the encrypted word. By default Midas allows Full Read Access to all the Midas Web pages. Only when modification of a Midas field the web password will be requested. The password is stored as a cookie in the target web client for 24 hours See **ODB /Experiment Tree**(p. ??).
- Other options of protection are described in **ODB /Experiment Tree**(p. ??) 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 HTML editor. By including custom Midas Tags, you



will have access to any field in the ODB of your experiment as well as the standard button for start/stop and page switch. See **mhttpd task(p.??)** , **Custom page(p.??)**.

5. **How do I prevent user to modify ODB values while the run is in progress?**

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

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

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

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

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

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

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

- The alarm scheme works only under **ONLINE**. See **ODB /Run-Info Tree(p.??)** 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 ...**

- ...

Midas Code and Libraries(p.??) - Top(p.??) - Data format(p.??)

## 9.8 Components

### **Quick Start**(p. ??) - **Top**(p. ??) - **Internal features**(p. ??)

Midas system is based on a modular scheme that allows scalability and flexibility. 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 briefly described below but throughout the documentation more detailed information will be given regarding each of their capabilities. All these components are available from the "off-the-shelf" package. Basic components such as the **Buffer Manager**(p. ??), **Online Database**(p. ??), **Message System**(p. ??), **Run Control**(p. ??) 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**(p. ??). A general picture of the Midas system is displayed below. The following link is a similar image with more information **Midas Structure**.

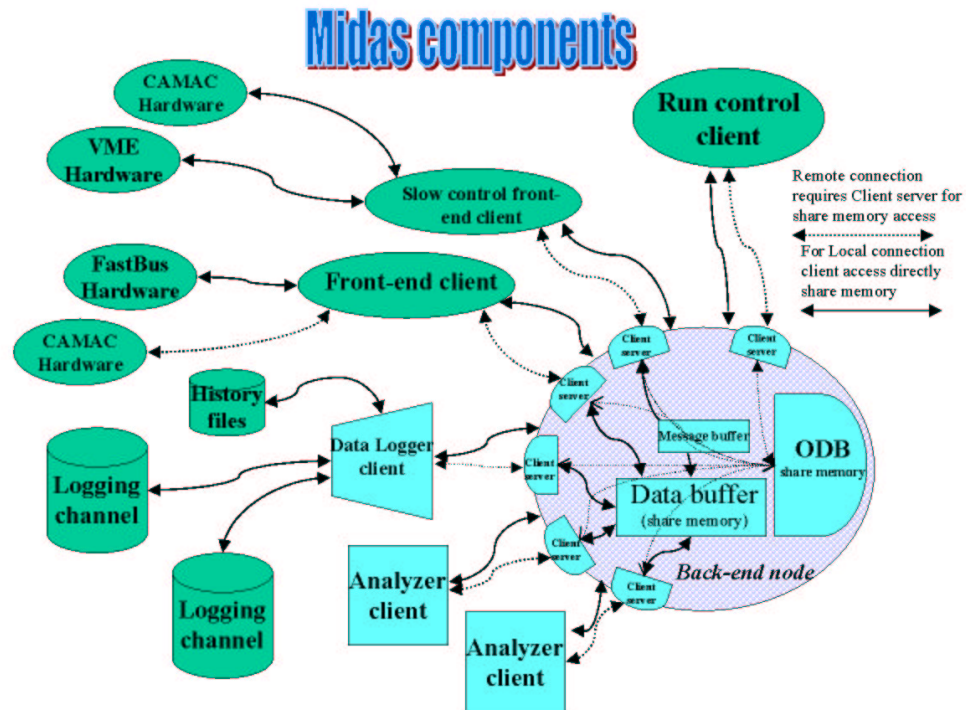


Figure 9.8: Components

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

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

- **Alarm System**(p. ??) Overall system and user alarm.
- **Electronic Logbook**(p. ??) Online User Logbook.

### 9.8.1 Buffer Manager

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

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

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

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

### 9.8.2 Message System

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

### 9.8.3 Online Database

In a distributed DAQ environment configuration data is usually stored in several files on different computers. MIDAS uses a different approach. All relevant

data for a particular experiment are stored in a central database called "Online Database" (ODB). This database contains run parameters, logging channel information, condition parameters for front-ends and analyzers and slow control values as well as status and performance data.

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

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

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

The Midas library provides a complete set of functions to manage and operate on these keys. Furthermore any ODB client can register a **Hot Link**(p.??) between a local C-structure and a element of the ODB. Whenever a client (program) changes a value in this sub-tree, 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**(p.??).

#### 9.8.4 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.

#### 9.8.5 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 identified event.

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

- Registers the given *Equipment(s)* list to the Midas system.
- Provides the mean of collecting the data from the hardware source defined 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 configuration.

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

- *Periodic events:* Scheduled event based on a fixed time interval. They can be used to read information such as scaler values, temperatures etc.
- *Polled events:* Hardware trigger information read continuously which in turn if signal is asserted will trigger the equipment readout.
- *Interrupt events:* Generated by particular hardware device supporting interrupt mode.
- *LAM events:* Generated only when pre-defined LAM is asserted:
- *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 files for hardware access to CAMAC, VME, Fastbus, GPIB and RS232 are part of Midas distribution set. For more information see **Frontend code**(p. ??).

### 9.8.6 Data Logger

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

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 Identifier.
- Auto restart feature allowing logging of several runs of a given size without user intervention.
- Recording of ODB values to a so called **History system(p.??)**
- Recording of the ODB to all or individual logging channel at the beginning and end of run state as well as to a separate disk file in a ASCII format. For more information see **ODB /Logger Tree(p.??)**.

### 9.8.7 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 is based on HBOOK histogramming functions from the CERN library which is not included in the MIDAS distribution. This analyzer takes care of receiving events (a few lines of code are necessary to receive events from the buffer manager), initializes the HBOOK system and automatically books N-tuples for all events. It calls user routines for event analysis.

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 file). When running online, generated N-tuples are stored in a ring-buffer in shared memory. They can be analyzed with PAW immediately without stopping the run.



When running off-line, the analyzer can read MIDAS binary files, analyze the events, add calculated data for each event and produce a HBOOK RZ output file which can be read in by PAW later. The analyzer framework also supports analyzer parameters. It automatically maps C-structures used in the analyzer to ODB records via **Hot Link**(p. ??). 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.

### 9.8.8 Run Control

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

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

In addition to these main transitions states, the *Tr\_Start* and *Tr\_Stop* have another 2 sub-states defined as *Tr\_preStart*, *Tr\_postStart*, *Tr\_preStop*, *Tr\_postStop*. These sub-states allows proper synchronization of data flow through the system.

#### Transitions

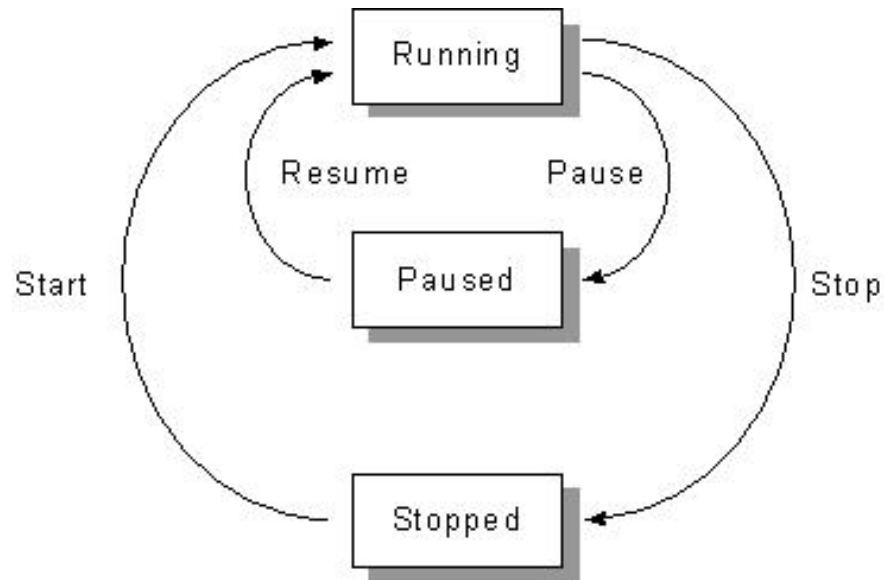


Figure 9.9: Transitions

### 9.8.9 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**(p.??) capability. Demand and measured values from slow control system equipment like high voltage power supplies or beam line magnets are stored directly in the ODB.

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

The Slow control system is organized in "Classes Driver ". Each Class driver refers to a particular set of functionality of that class i.e. High-Voltage, Temperature, General I/O, Magnet etc. The implementation of the device specific hardware is left to the hardware device driver. The current MIDAS distribution already has some device driver for general I/O and commercial High Voltage power supply system (see **Supported hardware**(p.??)). 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 NT/9x is already available. It can set, load and print high voltages for any devices of that class. For more information

see **Slow Control**(p. ??).

### 9.8.10 History system

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

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

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

### 9.8.11 Alarm System

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

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

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

- Logging the alarm message to the experiment log file.
- Sending a "Electronic Log message" (see **Electronic Logbook**(p. ??)).
- Interrupt data acquisition. For more information see **Alarm System**(p. ??), **ODB /Alarms Tree**(p. ??).

### 9.8.12 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**(p. ??) is running in the background of the system. For more information see **Electronic Logbook**(p. ??), **mhttpd task**(p. ??).

**Quick Start**(p. ??) - **Top**(p. ??) - **Internal features**(p. ??)

## 9.9 Event Builder Functions

Each frontend channel needs to send its "midas event" (fragment in this case) to a dedicated midas buffer with a unique "Event Identifier". This is specified within the frontend code in the equipment definition ( BUF1 instead of default SYSTEM, see also **ODB /Equipment Tree**(p.??)):

```
EQUIPMENT equipment[] = {
    { "Trigger1",          // equipment name
      1, 0,                // event ID, trigger mask
      "BUF1",              // event buffer
      ...
    }
```

The user has the possibility of interfering into the event building process at several stages:

**Begin of run** Like in the frontend, a hook to the begin of run is available for initialization etc.

**End of run** Like in the frontend, a hook to the end of run is available for proper closure of private task etc.

**Event-By-Event** Once all the fragments for a given serial number (match), the user has the possibility to access these fragments for further "user fragment analysis" and/or appending private data to the built event through the mean of bank creation (see **eb\_user()**(p.??)).

In the case of serial number mismatch or error returned from the user code, the event builder will abort by issuing a "stop run" transition and return to an "idle" state. **THERE IS NO RECOVERY PROCESS AVAILABLE YET!**

### 9.9.1 ODB/EBuilder Tree

The Event builder tree will be created with default settings from the mevb.h header file. The location of the tree is at the root level of the midas experiment. Each frontend fragment and the "built event" has its own subdirectory under the /EBuilder with a Settings tree defining the frontend characteristics and Statistics for status information.

The fields "Event ID", "Buffer" have to match the frontend equipment definition in order to guarantee the proper data transfer.

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

EBuilder	DIR						
Settings	DIR						
Event ID	WORD	1	2	35h	0	RWD	1
Trigger mask	WORD	1	2	35h	0	RWD	1
Buffer	STRING	1	32	35h	0	RWD	SYSTEM
Format	STRING	1	32	35h	0	RWD	MIDAS
User build	BOOL	1	4	35h	0	RWD	n
User Field	STRING	1	64	3h	0	RWD	1024
Event mask	DWORD	1	4	35h	0	RWD	3
Hostname	STRING	1	64	43m	0	RWD	dasdevpc
Statistics	DIR						
Events sent	DOUBLE	1	8	38m	0	RWD	1883
Events per sec.	DOUBLE	1	8	38m	0	RWD	0
kBytes per sec.	DOUBLE	1	8	38m	0	RWD	0
Channels	DIR						
Frag1	DIR						
Settings	DIR						
Event ID	WORD	1	2	35h	0	RWD	1
Trigger mask	WORD	1	2	35h	0	RWD	65535
Buffer	STRING	1	32	35h	0	RWD	BUF1
Format	STRING	1	32	35h	0	RWD	MIDAS
Event mask	DWORD	1	4	35h	0	RWD	1
Statistics	DIR						
Events sent	DOUBLE	1	8	38m	0	RWD	1883
Events per sec.	DOUBLE	1	8	38m	0	RWD	1881.12
kBytes per sec.	DOUBLE	1	8	38m	0	RWD	0
Frag2	DIR						
Settings	DIR						
Event ID	WORD	1	2	35h	0	RWD	2
Trigger mask	WORD	1	2	35h	0	RWD	65535
Buffer	STRING	1	32	35h	0	RWD	BUF2
Format	STRING	1	32	35h	0	RWD	MIDAS
Event mask	DWORD	1	4	35h	0	RWD	2
Statistics	DIR						
Events sent	DOUBLE	1	8	38m	0	RWD	1884
Events per sec.	DOUBLE	1	8	38m	0	RWD	1882.12
kBytes per sec.	DOUBLE	1	8	38m	0	RWD	0

## 9.9.2 EB Operation

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

- Build the mevb task:

```
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 fe1 \
    fe1.c camacnul.o /usr/local/lib/mfe.o /usr/local/lib/libmidas.a \
    -lm -lz -lutil -lnsl
```

```
cc -g -I/usr/local/include -I../drivers -DOS_LINUX -Dextname -o fe2 \
    fe2.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 different windows connecting to a defined experiment. – If no experiment defined yet, set the environment variable MIDAS\_DIR to your current directory before spawning the windows.

```
eb> pwd
/home/midas/midas-1.8.3/examples/eventbuilder
eb> setenv MIDAS_DIR /home/midas/midas-1.8.3/examples/eventbuilder
eb> odbedit
[local:Default:S]/>ls
System
Programs
Experiment
Logger
Runinfo
Alarms
[local:Default:S]/>q
eb>

xterm1: eb> fe1
xterm2: eb> fe2
xterm3: eb> mevb
xterm4: eb> odbedit

[local:Default:S]/>ls
System
Programs
Experiment
Logger
Runinfo
Alarms
Equipment
EBuilder
[local:Default:S]/>scl
Name          Host
Fe1           dasdevpc    <--- Frontend fragment 1
Fe2           dasdevpc    <--- Frontend fragment 2
EBuilder      dasdevpc    <--- Event builder
ODBEdit       dasdevpc

[local:Default:S]/>
[local:Default:S]/>start now
Starting run #2

12:12:11 [ODBEdit] Run #2 started
[local:Default:R]/>stop

12:12:13 [ODBEdit] Run #2 stopped
12:12:16 [EBuilder] Run 2 Stop on frag#0; events_sent 144; npulser 0
12:12:16 [EBuilder] Run 2 Stop on frag#1; events_sent 144; npulser 0
[local:Default:S]/>
```

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

```
Run 2
In eb_begin_of_run
nfrag : 2
bm_empty_buffer:1
bm_empty_buffer:1
Event Serial1 Fragment#:1 Data size:56 Serial1 Fragment#:2 Data size:56 Serial1
Event Serial2 Fragment#:1 Data size:56 Serial2 Fragment#:2 Data size:56 Serial2
Event Serial3 Fragment#:1 Data size:56 Serial3 Fragment#:2 Data size:56 Serial3
Event Serial4 Fragment#:1 Data size:56 Serial4 Fragment#:2 Data size:56 Serial4
Event Serial5 Fragment#:1 Data size:56 Serial5 Fragment#:2 Data size:56 Serial5
...
Event Serial141 Fragment#:1 Data size:56 Serial141 Fragment#:2 Data size:56 Serial141
Event Serial142 Fragment#:1 Data size:56 Serial142 Fragment#:2 Data size:56 Serial142
Event Serial143 Fragment#:1 Data size:56 Serial143 Fragment#:2 Data size:56 Serial143
Event Serial144 Fragment#:1 Data size:56 Serial144 Fragment#:2 Data size:56 Serial144
In eb_end_of_run
Run 2 Stop on frag#0; events_sent 144; npulser 0
Time between request and actual stop: 3457 ms
In eb_end_of_run
Run 2 Stop on frag#1; events_sent 144; npulser 0
Time between request and actual stop: 3459 ms
```

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

```
eb> odb -e midas
[local:midas:S]/>sc1
Name          Host
Fe1            mid001.triumf.ca      <-- Node 1
Fe2            mid002.triumf.ca      <-- Node 2
EBuilder       dasdevpc          <-- Node 3
ODBEdit        dasdevpc          <-- Node 3
[local:midas:S]/>

Thu> mevb -e midas
Program mevb/EBuilder version 2 started

New Run 209
In eb_begin_of_run
nfrag : 2
bm_empty_buffer:1
bm_empty_buffer:1
Event Serial1 Fragment#:1 Data size:56 Serial1 Fragment#:2 Data size:56 Serial1
Event Serial2 Fragment#:1 Data size:56 Serial2 Fragment#:2 Data size:56 Serial2
Event Serial3 Fragment#:1 Data size:56 Serial3 Fragment#:2 Data size:56 Serial3
Event Serial4 Fragment#:1 Data size:56 Serial4 Fragment#:2 Data size:56 Serial4
Event Serial5 Fragment#:1 Data size:56 Serial5 Fragment#:2 Data size:56 Serial5
...
Event Serial233 Fragment#:1 Data size:56 Serial233 Fragment#:2 Data size:56 Serial233
Event Serial234 Fragment#:1 Data size:56 Serial234 Fragment#:2 Data size:56 Serial234
Event Serial235 Fragment#:1 Data size:56 Serial235 Fragment#:2 Data size:56 Serial235
In eb_end_of_run
Run 209 Stop on frag#0; events_sent 235; npulser 0
Time between request and actual stop: 4488 ms
```



### 9.9.3 mevb Status/Bugs

- Jan 17/2002    – Initial Version composed of: Makefile : Build fe1, fe2, mevb. fe1.c : frontend code for event fragment 1. fe2.c : frontend code for event fragment 2. mevb.h : Event builder header file. **mevb.c**(p.??) : Event builder core code. **ebuser.c**(p.??) : User code for event building.
- Under Linux if the FEx are remote and have already collected data without the mevb running, on the next run start, the mevb will exit with event mismatch. Make sure the mevb is started before the FEx or the FEx are "fresh" when mevb is launched. It appears when mevb is running under Windows this problem is not occurring.

## 9.10 Internal features

### Quick Start(p. ??) - Top(p. ??) - Utilities(p. ??)

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.

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

### 9.10.1 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(p. ??)** (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 file, others of a C file plus a header file. The file names usually refer

to the manufacturer abbreviation followed by the model number of the device. The libraries are continuously expanding to widen Midas support.

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

The user frontend code consists of a 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**(p.??) value can be modified to reflect the purpose of the code.
  - **frontend\_call\_loop**(p.??)() enable the function **frontend\_loop**(p.??) to be run on every frontend loop.
  - **display\_period**(p.??) defined in millisecond the time interval between refresh of a frontend status display. The value of zero disable the display. If the frontend is started in the background with the display enabled, the stdout should be redirected to the null device to prevent process to hang.
  - **max\_event\_size**(p.??) specify the maximum size of the expected event in byte.
  - **event\_buffer\_size**(p.??) specify the maximum size of the buffer in byte to be allocated by the system. After these system parameters, the user may add his or her own declarations.

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

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

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

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

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

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

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

```

#define N_SCLR 8

CAMAC crate and slots
#define CRATE      0
#define SLOT_C212 23
#define SLOT_ADC   1
#define SLOT_TDC   2
#define SLOT_SCLR  3

```

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

```

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

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

```

- **[Remark]** Each equipment has the option to force it-self to run at individual transition time see **The Equipment structure(p.??) / RO\_RUNNING** . At transition time the system functions **begin\_of\_run(p.??)**, **end\_of\_run(p.??)**, **pause\_run(p.??)**, **resume\_run(p.??)** 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.
- **[Equipment definition]** See **The Equipment structure(p.??)** 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
,
...

```

- **[Pre-defined functions]** The sequence of function calls throughout the frontend code is the following:

```

frontend_init()(p.??), begin_of_run()(p.??),
[pause_run()(p.??)/resume_run()], end_of_run()(p.??) ,
frontend_exit()(p.??)

```

**frontend\_init()(p.??)** 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()(p.??)** This function is called for every run start transition. Allows to update user parameter, load/setup/clear hardware. At the exit of this function the acquisition should be armed and ready to test the LAM. In case of CAMAC frontend, the LAM has to be declared to the Crate Controller. The function **{cam\_lam\_enable(CRATE, SLOT\_IO)}** is then necessary in order to enable the proper LAM source station. The LAM source station has to also be enabled (F26).

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

```

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

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

cam_inhibit_clear(CRATE);              // Remove I

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

```

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

**poll\_event()**(p.??) If the equipment definition is {**EQ\_POLLED** as a acquisition type, the {**poll\_event** will be call as often as possible over the corresponding poll time (ex:500ms see **The Equipment structure**(p.??) ) 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 equipment. If the LAM is ORed for several station and is independent of the equipment, the LAM test can be simplified (see example below)

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

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

    }

    return 0;
}
```

- **[Remark]** When multiple LAM 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**(p.??) , **MIDAS event construction**(p.??) , **YBOS event construction**(p.??) .

```
// 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, "ADCO", TID_WORD, &pdata);
    ...
}
```

`pause / resume_of_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` file.

`end_of_run()`(p.??) 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` file.

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

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

`frontend_exit()`(p.??) This function runs when the frontend is requested to terminate. Can be used for local statistic collection etc.

### 9.10.1.1 The Equipment structure

To write a frontend program, the user section (frontend.c) has to have an equipment list organized as a structure definition. Here is the structure 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(0,0xFFFFF), // 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
    },
    ...
}
```

- **["trigger","scaler"]**: Each equipment has to have a unique equipment name defined under a given node. The name will be the reference name of the equipment generating the event.
- **[1, 0]**: Each equipment has to be associated to 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 modified dynamically by the readout routine to define a sub-event type on an event-by-event basis. This can be used to mix "physics events" (from a physics trigger) and "calibration events" (from a clock for example) in one run and identify them later. Both parameters are declared as 16bit value. If the Trigger mask is used in a single bit-wise mode, only up to 16 masks are possible.
- **["SYSTEM"]** After composition of an "equipment", the Midas frontend **mfe.c**(p. ??) takes over the sending of this event to the "system buffer" on the back-end computer. Dedicated buffer can be specified on those lines allowing a secondary stage on the back-end (Event builder to collect and assemble these events coming from different buffers in order to compose a larger event. In this case the 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 field is left empty ("") the readout function associated to that equipment will still be performed, but the actual event won't be sent to the buffer. The positive side-effect of that configuration is to allow that particular equipment to be mirrored in the ODB if the RO\_ODB is turned on.

- **[EQ\_xxx]** The field specify the type of equipment. It can be composed of several bitwise flags. The following **EQ\_POLLED(p.??)**, **EQ\_INTERRUPT(p.??)** and **EQ\_SLOW(p.??)** flags cannot be Ored together. The possible options are:

**EQ\_POLLED(p.??)** In this mode, the name of the routine performing the trigger check function is defaulted to **poll\_event()**(p.??). As polling consists on checking a variable for a true condition, if the loop would be infinite, 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 flag is mainly used for data acquisition based on a "LAM".

```
EQUIPMENT equipment[] = {
    { "Trigger",          // equipment name    ...
      500,                // poll for 500ms
      ...
    }
```

**EQ\_INTERRUPT(p.??)** For this mode, Midas requires complete configuration and control of the interrupt source. This is provided by an interrupt configuration routine **interrupt\_configure()**(p.??) that has to be coded by the user in the user section of the frontend code. A pointer to this routine is passed to the system instead of the polling routine. The interrupt configuration routine has the following declaration:

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

```
return CM_SUCCESS;
```

**EQ\_PERIODIC**(p.??) In this mode the function associated to this equipment is called periodically. No hardware requirements is necessary to trigger the read-out function. The "poll" field in the equipment declaration is in this case used for periodicity.

**EQ\_SLOW**(p.??) 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**(p.??) This flag enables the equipment to be triggered by remote procedure call (RPC). If present, the web interface will provide a button for that action.

**EQ\_FRAGMENTED**(p.??) This flag enables large event (beyond Midas configuration limit) to be handled by the system. This flag requires to have a valid **max\_event\_size\_frag** variable defined in the user frontend code (frontend.c). The **max\_event\_size**(p.??) 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.

**LAM\_SOURCE**(p.??)(0,0xFFFFFFFF) This parameter is a bit-wise representation of the 24 CAMAC slots which may raise the LAM. It defines which CAMAC slot is allowed to trigger the call to the readout routine. (See `read_trigger_event()` ).

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

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

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

**RO\_RUNNING(p.??)** Specify when the read-out of an event should be occurring (transition state) or be enabled (state). Following options are possible:

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

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

500 Time interval for Periodic equipment (EQ\_PERIODIC) or time out value in case of EQ\_POLLING (unit in millisecond).

0 (stop after...) Specify the number of events to be taken prior forcing an End-Of-Run transition. The value 0 disables this option.

0 (**Super Event(p.??)**) Enable the Super event capability. Specify the maximum number of events in the Super event.

0 (**History system(p.??)**) Enable the MIDAS history system for that equipment. The value (positive in seconds) indicates the time interval for generating the event to be available for history logging by the mlogger task if running.

"" , "" , "" Reserved field for system. Should be present and remain empty.

read\_trigger\_event() User read-out routine declaration (could be any name). Every time the frontend is initialized, it copies the equipment settings to the ODB under /Equipment/<equipment name>/Common. A hot-link to that ODB tree is created allowing some of the settings to be changed during run-time. Modification of "Enabled" flag, RO\_xxx flags, "period" and "event limit" from the ODB is immediately reflected into the frontend which will act

upon them. This function has to be present in the frontend code and will be called for every trigger under one of the two conditions:

In polling mode The poll\_event has detected a trigger request while polling on a trigger source.

In interrupt mode An interrupt source pre-defined through the interrupt\_configuration has occurred.

Remark 1 The first argument of the readout function provide the pointer to the newly constructed event and point to the first 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.

### 9.10.1.2 FIXED event construction

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

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

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

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

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

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

```

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

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

    return sizeof(TRIGGER_EVENT);
}

```

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

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

- **bk\_init()**(p.??) , **bk\_init32()**(p.??) Initializes a bank structure in an event.
- **bk\_create()**(p.??) Creates a bank with a given name (exactly four characters)
- **bk\_close()**(p.??) Closes a bank previously opened with **bk\_create()**(p.??).
- **bk\_locate()**(p.??) Locate a bank within an event by its name.
- **bk\_iterate()**(p.??) Return bank and data pointers to each bank in the event.
- **bk\_list()**(p.??) Construct a string of all the bank name in the event.
- **bk\_size()**(p.??) Returns the size in bytes of all banks including the bank headers in an event. The following code composes an event containing two

ADC and two TDC values, the <...> statements have to be filled with specific code accessing the hardware:

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

    bk_init(pevent);

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

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

    return bk_size(pevent);
}
```

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

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

    // init bank structure
    bk_init(pevent);

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

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

    bk_close(pevent, pdata);

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

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

    bk_close(pevent, pdata);

    return bk_size(pevent);
}
```

### 9.10.3 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()**(p.??) Initializes a bank structure in an event.
- **ybk\_create()**(p.??) Creates a bank with a given name (exactly four characters)
- **ybk\_close()**(p.??) Closes a bank previously opened with **ybk\_create()**(p.??).
- **ybk\_size()**(p.??) 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));
}
```



### 9.10.4 Deferred Transition

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

- It may not be advised to pause or stop a run until let say some hardware has turned off a particular valve.
- The start of the acquisition system is 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 satisfied. The Deferred\_Transition requires 3 steps for setup:

- Register the deferred transition.

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

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

- Provide callback function to serve the deferred transition

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

        if (end_of_mcs_cycle)
        {
            transition_PS_requested = FALSE;
            end_of_mcs_cycle = FALSE;
            return TRUE;
        }
    }
    else
        return FALSE;
}
```

- Implement the condition code

```

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

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

        re_arm_cycle();
        return bk_size(pevent);
    }
}

```

In the example above the frontend code register for PAUSE and STOP. The second argument of the `cm_register` *wait\_end\_cycle* is the declaration of the callback function. The callback function will be called as soon as the transition is requested and will provide the Boolean flag first to be TRUE. By setting the *transition\_PS\_requested*, the user will have the acknowledgment of the transition request. By returning FALSE from the callback you will prevent the transition to occur. As soon as the user condition is satisfied (`end_of_mcs_cycle = TRUE`), the return code in the callback will be set to TRUE and the requested transition will be issued. The Deferred transition shows up in the ODB under **/run-info/Requested transition** and will contain the transition code (see **State Codes & Transition Codes**(p.??) ). 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 override will do the transition function regardless of the state of the hardware involved.

### 9.10.5 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()`(p.??), 20 bytes for `bk_init32()`(p.??) and `ybk_init()`(p.??). If the data section size is close to the number above, the data transfer as well as the data storage has an non-negligible overhead. To address this problem, the equipment can be setup to generate a

so called *Super* Event which is an event composed of the initial standard bank header for the first event on the super event and up to **number** of sub event maximum successive data section before closing of the bank.

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

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

```
{ "GE",                // equipment name
  2, 0x0002,           // event ID, trigger mask
  "SYSTEM",            // event buffer
#ifdef USE_INT
  EQ_INTERRUPT,        // equipment type
#else
  EQ_POLLED,           // equipment type
#endif
  LAM_SOURCE(GE_C, LAM_STATION(GE_N)), // event source
  "MIDAS",             // format
  TRUE,                // enabled
  RO_RUNNING,          // read only when running
  200,                 // poll for 200ms
  0,                   // stop run after this event limit
  1000,                // ----> number of sub event <----- enable Super event
  0,                   // don't log history
  "", "", "",          //
  read_ge_event,       // readout routine
  ,
  ...
```

- Setup the readout function for Super Event collection.

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

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

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

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

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

```

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

// clear hardware
re_arm_ge();

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

    return NWORDS * sizeof(WORD) + sizeof(BANK_HEADER) + sizeof(BANK);
}
else
    // Return the data section size only
    // sizeof(WORD) is defined by the TID_WORD in bk_create()

    return NWORDS * sizeof(WORD);

```

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

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

### 9.10.6 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 modified, the frontend receives a notification and sets the new value. In the other direction the frontend continuously reads the voltage and current values from all channels and updates the according ODB arrays if there has been a significant change. This design has a possible inconvenience due to 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

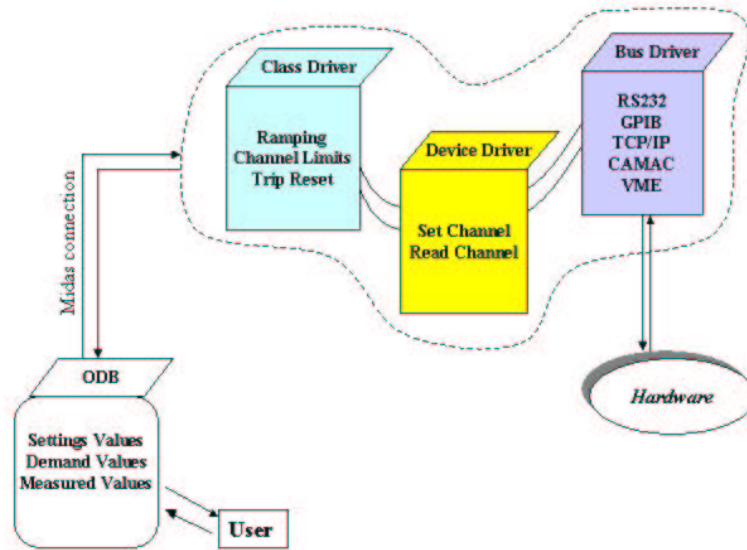


Figure 9.10: Class driver, Device and Bus driver in the slow control system

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

Key name	Type	#Val	Size	Last	Opn	Mode	Value
-----							
Epics	DIR						
Settings	DIR						
Channels	DIR						
Epics	INT	1	4	25h	0	RWD	3
Devices	DIR						
Epics	DIR						
Channel name	STRING	10	32	25h	0	RWD	
		[0]				GPS:VAR1	

				[1]			GPS:VAR2
				[2]			GPS:VAR3
Names	STRING	10	32	17h	1	RWD	
				[0]			Current
				[1]			Voltage
				[2]			Watchdog
Update Threshold Measure	FLOAT	10	4	17h	0	RWD	
				[0]			2
				[1]			2
				[2]			2
Common	DIR						
Event ID	WORD	1	2	17h	0	RWD	3
Trigger mask	WORD	1	2	17h	0	RWD	0
Buffer	STRING	1	32	17h	0	RWD	SYSTEM
Type	INT	1	4	17h	0	RWD	4
Source	INT	1	4	17h	0	RWD	0
Format	STRING	1	8	17h	0	RWD	FIXED
Enabled	BOOL	1	4	17h	0	RWD	y
Read on	INT	1	4	17h	0	RWD	121
Period	INT	1	4	17h	0	RWD	60000
Event limit	DOUBLE	1	8	17h	0	RWD	0
Num subevents	DWORD	1	4	17h	0	RWD	0
Log history	INT	1	4	17h	0	RWD	1
Frontend host	STRING	1	32	17h	0	RWD	hostname
Frontend name	STRING	1	32	17h	0	RWD	Epics
Frontend file name	STRING	1	256	17h	0	RWD	feepic.c
Variables	DIR						
Demand	FLOAT	10	4	0s	1	RWD	
				[0]			1.56
				[1]			120
				[2]			87
Measured	FLOAT	10	4	2s	0	RWD	
				[0]			1.56
				[1]			120
				[2]			87
Statistics	DIR						
Events sent	DOUBLE	1	8	17h	0	RWDE	26
Events per sec.	DOUBLE	1	8	17h	0	RWDE	0
kBytes per sec.	DOUBLE	1	8	17h	0	RWDE	0

### 9.10.7 Electronic Logbook

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

### 9.10.8 Log file

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

without **ODB /Logger Tree**(p.??) In this case the location is defined by either the **MIDAS\_DIR**(p.??) environment (see **Environment variables**(p.??) ) or the definition of the experiment in the **exptab** file (see **Experiment\_Definition** ). In both case the log file will be in the experiment specific directory.

with **/Logger Tree** The **midas.log** will be sitting into the defined directory specified by **Data Dir** .

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

The **MIDAS Macros**(p.??) definition provides a list of possible type of messages.

```
Fri Mar 24 10:48:40 2000 [CHAOS] Run 8362 started
Fri Mar 24 10:48:40 2000 [Logger] Run #8362 started
Fri Mar 24 10:55:04 2000 [Lazy_Tape] cni-043[10] (cp:383.6s) /dev/nst0/run08360.ybs 849.896MB file NEW
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 NEW
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 [ODBEEdit] 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**(p.??) - **Top**(p.??) - **Utilities**(p.??)



## 9.11 Introduction

**Top(p.??) - Top(p.??) - New Documented Features(p.??)**

*... A few words...*

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

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

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

### 9.11.1 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).

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 devel-

opment involves RTLinux for either dedicated frontend or composite frontend and backend system.

For the newest status, check the MIDAS home page: [Switzerland](#) , [Canada](#)

### 9.11.2 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.

**Top(p. ??) - Top(p. ??) - New Documented Features(p. ??)**

## 9.12 mhttpd task

**Utilities(p. ??) - Top(p. ??) - Data format(p. ??)**

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

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

The ...

- **Start page(p. ??)**
- **ODB page(p. ??)**
- **Equipment page(p. ??)**
- **CNAF page(p. ??)**
- **Message page(p. ??)**
- **Elog page(p. ??)**

- **Program page**(p. ??)
- **History page**(p. ??)
- **Alarm page**(p. ??)
- **Custom page**(p. ??)

**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-divided in several sections:

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

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

**Start... button** Depending on the run state, a single or the two first buttons will be showing the possible action (Start/Pause/Resume/Stop) (see **Start page**(p. ??)).

**ODB button** Online DataBase access. Depending on the security, R/W access can be granted to operated on any ODB field (see **ODB page**(p. ??)).

**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**(p. ??)).

**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**(p. ??)).

**Elog button** Electronic Log book. Permit to record permanently (file) comments/messages composed by the user (see **Elog page**(p. ??)).

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**(p.??))

Program button Display current program (midas application) status. Each program has a specific information record associated to it. This record is tagged as a hyperlink in the listing (see **Program page**(p.??)).

History button Display History graphs of pre-defined variables. The history setting has to be done through ODB under the /**History** (see **History system**(p.??), **History page**(p.??)).

Config button Allows to change the page refresh rate.

Help button Help and link to the main Midas web pages.

User button(s) If the user define a new tree in ODB named **Script** than any sub-tree name will appear as a button of that name. Each sub-tree (/Script/<button name>/) should contain at least one string key being the script command to be executed. Further keys will be passed as

- **Arguments** to the script. Midas Symbolic link are permitted.
- **Example** : The **Example** below defines a script names **doit** with 2 **Arguments** (run# device) which will be invoked when the button <doit> is pressed.

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

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

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

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

```

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

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

```

General info Current run number, state, General Enable flag for Alarm, Auto restart flag Condition of mlogger.

Equipment listing Equipment name (see **Equipment page**(p.??)), 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**(p.??) is running.

Lazylogger listing Lazylogger list. Multiple lazy application can be active. This section is present only when the Midas application **lazylogger task**(p.??) is running.

Last system message Display a single line containing the last system message received at the time of the last display refresh.

Current client listing List of the current active Midas application with the hostname on which their running.

Midas Web server

The screenshot displays the MIDAS Web server interface with the following sections and data:

- Title:** MIDAS experiment "midas" | Mon Dec 18 14:42:06 2000
- Action/Pages:** Start | ODB | CNAF | Messages | ELog | Alarms | Programs | History | Config | Help
- User button(s):** doit | doit2
- Trigger button(s):** Trigger Scaler event
- Alias/Alias new window:** Trng setting doit setting
- General Info:**

Run #63	Stopped	Alarms On	Perfom Test	Logging disabled
Start: Wed Nov 22 10:00:37 2000		Stop: Wed Nov 22 10:01:48 2000		
- Equipment listing:**

Equipment	FE Node	Events	Event rate[/s]	Data rate[kB/s]	Analyzed
Trigger	feflash@midmes04	7111	0.0	0.0	73.9%
Scaler	feflash@midmes04	0	0.0	0.0	0.0%
- Logger Channels:**

Channel	Active	Events	MB written	GB total
0 run00063.mid	Disabled	0	0.000	0.000
1 run00063.mid	Disabled	0	0.000	0.000
- Lazylogger application:**

Lazy Label	Progress	File Name	# Files	Total
Disk_01	0 %		0	0.0 %
Tape_01	0 %		0	0.0 %
- Last system message:** Mon Dec 18 14:40:06 2000 [mhttpd] Program mhttpd on host midmes04 started
- Client listing:**

feflash [midmes04]	Logger [midmes04]	Lazy_Disk [midmes04]
Lazy_Tape [midmes04]	mhttpd [midmes04]	

Figure 9.11: Midas Web server

### 9.12.1 Start page

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

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

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

Figure 9.12: Start run request page.



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

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

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

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

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

MIDAS experiment "midas"		Fri Oct 12 10:33:15 2001	
Start new run			
Run number	<input type="text" value="2"/>		
Write Data	<input type="text" value="y"/>		
Enable logging	<input type="text" value="n"/>		
enable	<input type="text" value="n"/>		
Scaler for expt B1 only	<input type="text" value="0"/>		
number of channels	<input type="text" value="0"/>		
<i>maximum 1024</i>			
dwelling time (ns)	<input type="text" value="0"/>		
<b>Check hardware now</b>			
<input type="button" value="Start"/> <input type="button" value="Cancel"/>			

Figure 9.13: Start run request page.

### 9.12.2 ODB page

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

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

ODB page access.

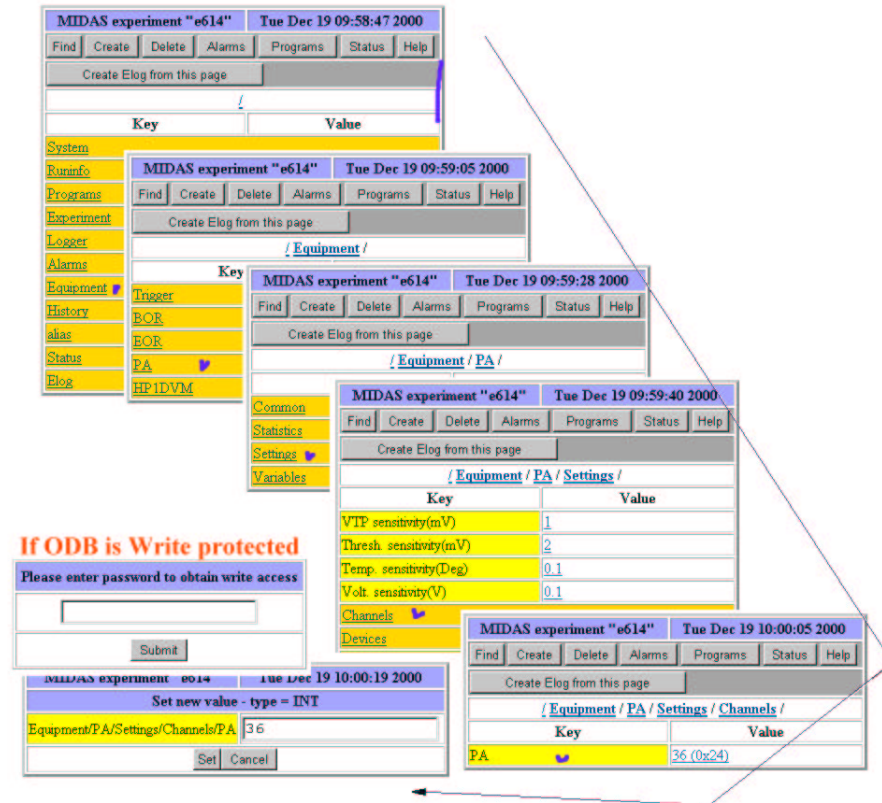


Figure 9.14: ODB page access.

### 9.12.3 Equipment page

The equipment names are linked to their respective /**Variables** sub-tree. This permit to access as a shortcut the current values of the equipment. In the

case the equipment is a slow control equipment, the parameters list may be hyperlinked for parameter modification. This option is possible only if the parameter names have a particular name syntax (see **History system**(p. ??)).

Slow control page.

MIDAS experiment "e614"						Mon Dec 18 14:21:54 2000				
<div><div>ODB</div><div>Status</div><div>Help</div></div>										
Equipment: PA										
Groups: <a href="#">All</a> <a href="#">Crate0</a> <a href="#">Crate1</a>										
Names	D_VTp	M_VTp	D_Thres	M_ThresA	M_ThresB	D_TP	M_TP	Temp	Voltage+	Voltage-
SI_0	<a href="#">0</a>	0	<a href="#">0</a>	0	0	<a href="#">n</a>	n	51	-0.018	-0.006
SI_1	<a href="#">1850</a>	1852	<a href="#">1011</a>	-1002	-998	<a href="#">n</a>	n	31.3	5.061	-5.103
SI_2	<a href="#">1793</a>	1793	<a href="#">1017</a>	-1002	-999	<a href="#">n</a>	n	33.8	5.099	-5.112
SI_3	<a href="#">1775</a>	1774	<a href="#">1023</a>	-1001	-1000	<a href="#">n</a>	n	33.5	5.067	-5.093
SI_4	<a href="#">1852</a>	1852	<a href="#">1017</a>	-1003	-999	<a href="#">n</a>	n	34.9	5.076	-5.104
SI_5	<a href="#">1800</a>	1800	<a href="#">1014</a>	-1004	-1000	<a href="#">n</a>	n	38.5	5.055	-5.108
SI_6	<a href="#">1786</a>	1785	<a href="#">1011</a>	-1001	-1000	<a href="#">n</a>	n	40.4	5.066	-5.098
SI_7	<a href="#">1798</a>	1798	<a href="#">1011</a>	-1004	-1000	<a href="#">n</a>	n	37.3	5.083	-5.097
SI_8	<a href="#">1795</a>	1795	<a href="#">1018</a>	-1002	-1002	<a href="#">n</a>	n	32	5.073	-5.092
SI_9	<a href="#">1801</a>	1801	<a href="#">1016</a>	-1001	-1002	<a href="#">n</a>	n	35.1	5.09	-5.104
SI_10	<a href="#">1797</a>	1798	<a href="#">1033</a>	-1001	-1000	<a href="#">n</a>	n	34.7	5.065	-5.104
SI_11	<a href="#">1795</a>	1796	<a href="#">1019</a>	-1000	-1002	<a href="#">n</a>	n	31.3	5.057	-5.102
SI_12	<a href="#">1797</a>	0	<a href="#">1013</a>	0	0	<a href="#">n</a>	n	0	-0.022	-0.006
SI_13	<a href="#">1798</a>	1798	<a href="#">1016</a>	-1002	-1000	<a href="#">n</a>	n	34.3	5.067	-5.102
SI_14	<a href="#">1793</a>	1793	<a href="#">1016</a>	-1000	-1000	<a href="#">n</a>	n	32.4	5.07	-5.095
SI_15	<a href="#">1799</a>	1800	<a href="#">1015</a>	-1000	-1001	<a href="#">n</a>	n	28.9	5.068	-5.092
SI_16	<a href="#">1782</a>	1783	<a href="#">1007</a>	-1002	-1001	<a href="#">n</a>	n	37.7	5.058	-5.099
SI_17	<a href="#">1798</a>	1798	<a href="#">1011</a>	-1001	-999	<a href="#">n</a>	n	33.3	5.104	-5.094
SI_18	<a href="#">1796</a>	1796	<a href="#">1017</a>	-1001	-1002	<a href="#">n</a>	n	30.6	5.078	-5.103
SI_19	<a href="#">1798</a>	1797	<a href="#">1009</a>	-1000	-1001	<a href="#">n</a>	n	34.7	5.07	-5.106
SI_20	<a href="#">1803</a>	1803	<a href="#">1014</a>	-1002	-1000	<a href="#">n</a>	n	37.6	5.066	-5.11
SI_21	<a href="#">1799</a>	1799	<a href="#">1010</a>	-1000	-1002	<a href="#">n</a>	n	38.7	5.056	-5.11
SI_22	<a href="#">1805</a>	1805	<a href="#">1015</a>	-1000	-1001	<a href="#">n</a>	n	33.1	5.066	-5.114
SI_23	<a href="#">1793</a>	1793	<a href="#">1019</a>	-1000	-1001	<a href="#">n</a>	n	31.2	5.055	-5.096
SI_24	<a href="#">1789</a>	1788	<a href="#">1018</a>	-1000	-1002	<a href="#">n</a>	n	38.1	5.047	-5.105

Figure 9.15: Slow control page.

#### 9.12.4 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.

MIDAS experiment "silicon"			CAMAC server: feSilicon	
Execute			ODB Status Help	
N	A	F	Data	
1	0	0	0	
Repeat		1	C cycle Z cycle	
Repeat delay [ms]		0	Set inhibit Clear inhibit	
Data increment		0	Branch 0	
A increment		0	Crate 1	

MIDAS experiment "trinat"			No CAMAC server running	
Execute			ODB Status Help	
N	A	F	Data	
1	0	0	0	
Repeat		1	C cycle Z cycle	
Repeat delay [ms]		0	Set inhibit Clear inhibit	
Data increment		0	Branch 0	
A increment		0	Crate 1	

Figure 9.16: CAMAC command pages.

### 9.12.5 Message page

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

Message page.

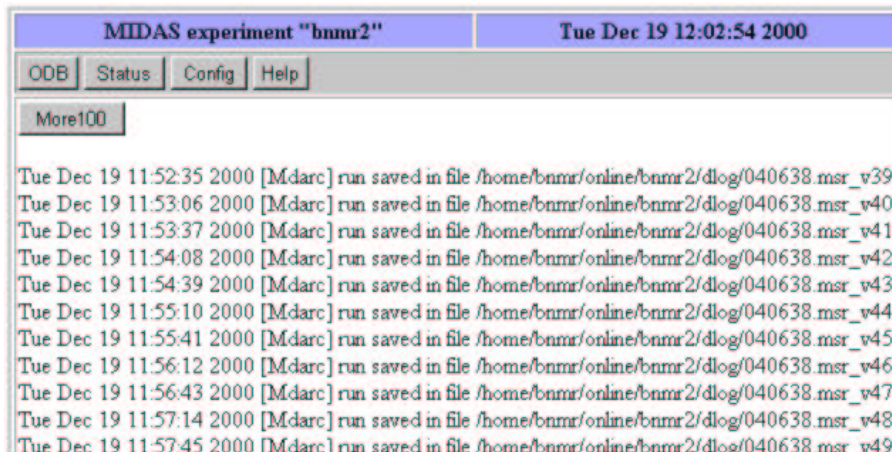


Figure 9.17: Message page.

### 9.12.6 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.

MIDAS Electronic Logbook		Experiment "chaos"	
New	Edit	Reply	Query
Last 10 entries		Shift Check	Runlog
Status			
Next	Previous	Last	Check a category to browse only entries from that category
Entry date: Sun Nov 19 06:10:20 2000		Run number: 13079	
<input type="checkbox"/> Author: rmeier		<input type="checkbox"/> Type: Shift Check	
<input type="checkbox"/> System: General		<input type="checkbox"/> Subject:	
<pre> 1 Log beam channel : [X] adjusted B1 (.5 Gauss) 2 Target T-P Ok? : [X] MT running 3 All Chambers V-I Ok? : [X] 4 DAQ : [X] 5 Histograms, dotplots Ok? : [X] </pre>			

Figure 9.18: main Elog page.

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

HTML Elog message.

MIDAS Electronic Logbook		Experiment "tuda"																											
New	Edit	Reply	Query																										
Last 24 hours		Runlog	Status																										
Next	Previous	Last	Check a category to browse only entries from that category																										
Entry date: Thu Sep 14 14:55:34 2000		Run number: 1																											
<input type="checkbox"/> Author: midas@midmes02.triumf.ca		<input type="checkbox"/> Type: Info																											
<input type="checkbox"/> System: General		<input type="checkbox"/> Subject: DAQ																											
<p><b>Hello TUDA folks,</b></p> <ul style="list-style-type: none"> <li>The main components of the DAQ for upcoming run is "basically" installed.</li> <li>The VME crates contains the PPC and the <a href="#">CES CBD8210</a> CAMAC branch driver.</li> <li>This CBD is connected to two A2 CAMAC Crate Controllers.</li> <li>Acquisition for 16x8 ADCs + 4x32 TDCs.</li> </ul> <table border="1"> <thead> <tr> <th>CRATE 1</th> <th>Modules</th> </tr> </thead> <tbody> <tr> <td>Slot 01-16</td> <td>ADC 4418 <a href="#">Siena</a></td> </tr> <tr> <td>Slot 17-20</td> <td>TDC 3377 <a href="#">LeCroy</a> or <a href="#">Command list</a></td> </tr> <tr> <td>Slot 21</td> <td>Output Register <a href="#">OR2027 SEN</a></td> </tr> <tr> <td>Slot 22-23</td> <td>Pattern Unit C212</td> </tr> <tr> <td>Slot 24-25</td> <td>Crate Controller A2 <a href="#">Iorway 71B Spec</a></td> </tr> <tr> <th>CRATE 2</th> <th>Modules</th> </tr> <tr> <td>Slot 01</td> <td>Hex 24bit Scalers KCS3815</td> </tr> <tr> <td>Slot 22-23</td> <td>Branch terminator BHT-002/D SEC</td> </tr> <tr> <td>Slot 24-25</td> <td>Crate Controller A2 1302 BtRa system</td> </tr> </tbody> </table> <p><b>System Status log:</b></p> <table border="1"> <thead> <tr> <th>Date</th> <th>Successful</th> <th>Unsuccessful or not done yet</th> </tr> </thead> <tbody> <tr> <td>September 14/2000</td> <td>Optical 100BaseT link to the Shack</td> <td></td> </tr> </tbody> </table>				CRATE 1	Modules	Slot 01-16	ADC 4418 <a href="#">Siena</a>	Slot 17-20	TDC 3377 <a href="#">LeCroy</a> or <a href="#">Command list</a>	Slot 21	Output Register <a href="#">OR2027 SEN</a>	Slot 22-23	Pattern Unit C212	Slot 24-25	Crate Controller A2 <a href="#">Iorway 71B Spec</a>	CRATE 2	Modules	Slot 01	Hex 24bit Scalers KCS3815	Slot 22-23	Branch terminator BHT-002/D SEC	Slot 24-25	Crate Controller A2 1302 BtRa system	Date	Successful	Unsuccessful or not done yet	September 14/2000	Optical 100BaseT link to the Shack	
CRATE 1	Modules																												
Slot 01-16	ADC 4418 <a href="#">Siena</a>																												
Slot 17-20	TDC 3377 <a href="#">LeCroy</a> or <a href="#">Command list</a>																												
Slot 21	Output Register <a href="#">OR2027 SEN</a>																												
Slot 22-23	Pattern Unit C212																												
Slot 24-25	Crate Controller A2 <a href="#">Iorway 71B Spec</a>																												
CRATE 2	Modules																												
Slot 01	Hex 24bit Scalers KCS3815																												
Slot 22-23	Branch terminator BHT-002/D SEC																												
Slot 24-25	Crate Controller A2 1302 BtRa system																												
Date	Successful	Unsuccessful or not done yet																											
September 14/2000	Optical 100BaseT link to the Shack																												

Figure 9.19: HTML Elog message.

The **runlog** button display the content of the file **runlog.txt** which is expected to be in the data directory specified by the ODB key **/Logger/Data Dir** . Regardless of its content, it will be displayed in the web page. Its common uses is to **append** lines after every run. The task appending this run information can be any of the midas application. **Example** is available in the **Exam-**



ple /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 "Itao"					
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 9.20: Elog page, Runlog display.

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

Elog page, New Elog entry form.

MIDAS Electronic Logbook		Experiment "chaos"	
<input type="button" value="Submit"/>			
Entry date: Tue Dec 19 12:09:13 2000		Run number: <input type="text" value="13397"/>	
Author: <input type="text"/>		Type: <input type="text" value="Routine"/>	
System: <input type="text" value="General"/>		Subject: <input type="text"/>	
<div> <div>General</div> <div>           DAO            Detector            Electronics            Target            Beamline         </div> </div>		<div> <div>Routine</div> <div>           Shift summary            Minor error            Severe error            Fix            Info            Modification            Complaints            Reply            Alarm            Test            Other         </div> </div>	
<div> <div>Text:</div> <div></div> </div>			
<input type="checkbox"/> Submit as HTML text			
Enter attachment filename(s) or ODB tree(s), use "\*" as an ODB directory separator.			
Attachment1: <input type="text"/>		<input data-bbox="1208 730 1312 756" type="button" value="Browse..."/>	
Attachment2: <input type="text"/>		<input data-bbox="1208 777 1312 802" type="button" value="Browse..."/>	
Attachment3: <input type="text"/>		<input data-bbox="1208 823 1312 848" type="button" value="Browse..."/>	

Figure 9.21: Elog page, New Elog entry form.

### 9.12.7 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 beginning or the end of a run.

Program page.



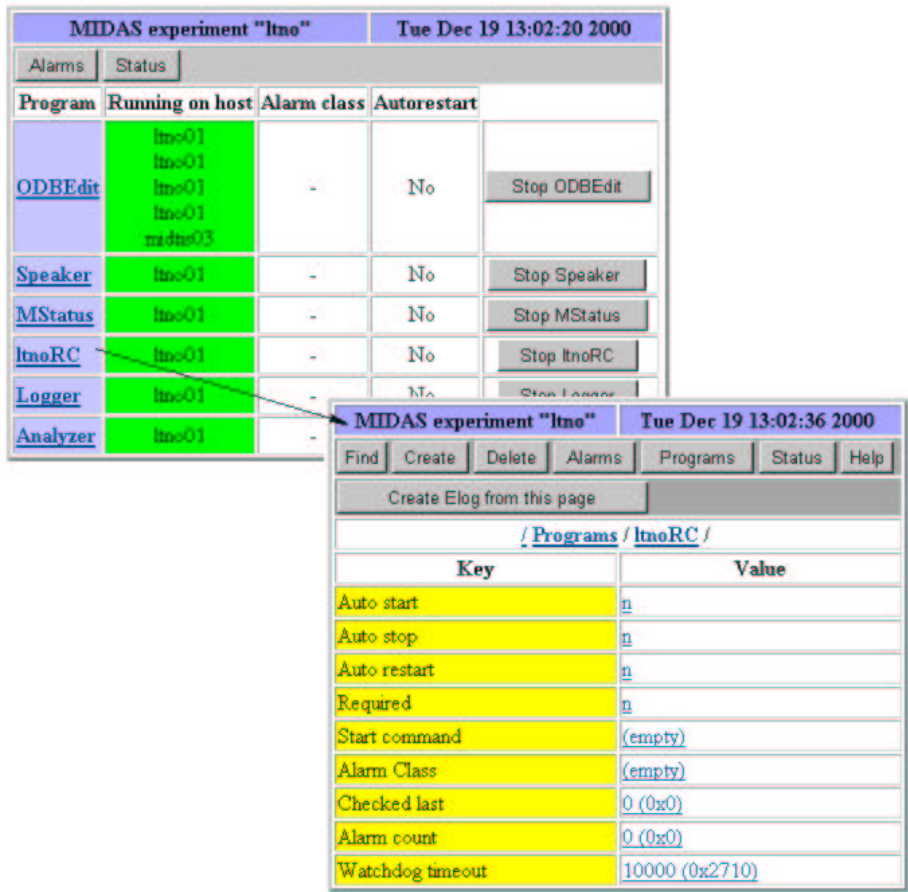


Figure 9.22: Program page.

### 9.12.8 History page

This page reflects the **History system**(p. ??) settings. It lists on the top of the page the possible pannels defined in the ODB. A serie of buttons defines the full time scale of the graph and the ">" "+" "-" ">" ">>" buttons permit the shifting of the graph in the time direction. The main graph will always display all the defined channels but clicking on the boxed channel names will supersede the page with this single channel. Elog message with the current history display as attachement for reference is possible using the **Elog** button.

From version 1.9.0, the history page display has been greatly improved:

- 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.

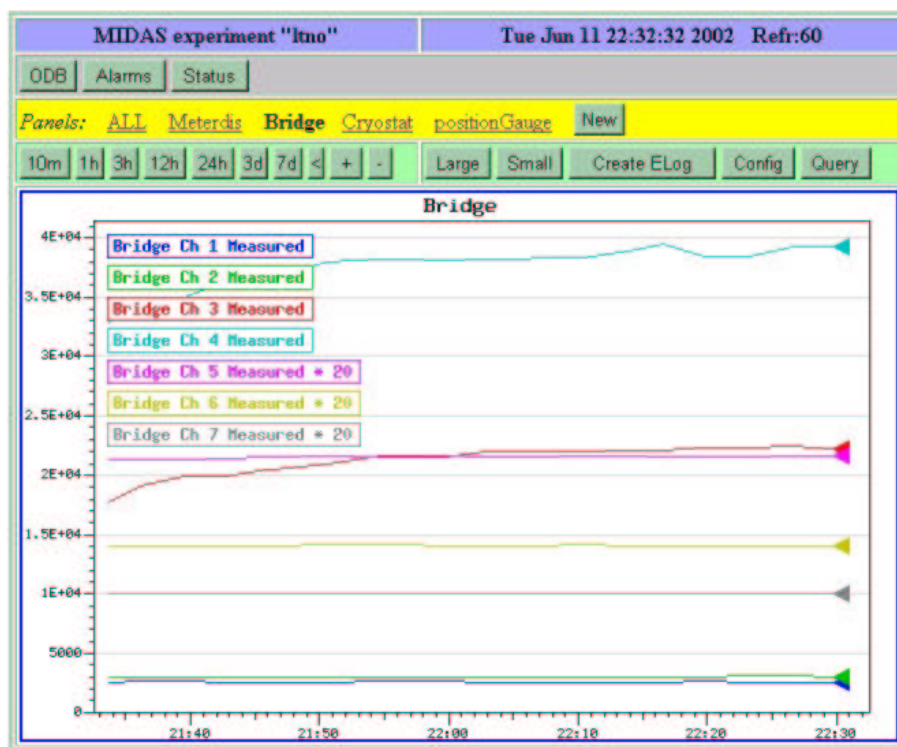


Figure 9.23: History page.

History channel selection Page.

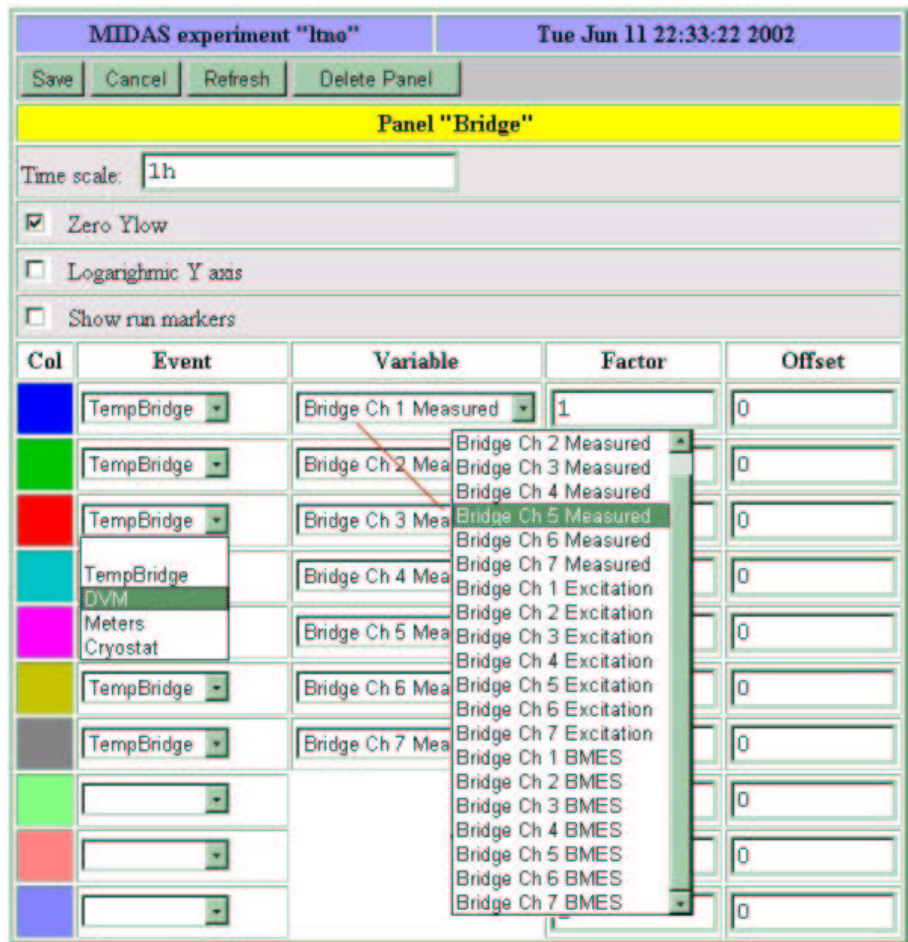


Figure 9.24: History channel selection Page.

### 9.12.9 Alarm page

This page reflects the **Alarm System**(p. ??) settings. It presents the four type of alarms:

- Evaluated alarms Triggered by ODB value on given arithmetical condition.
- Program alarms Triggered on condition of the state of the defined task.
- Internal alarms Trigger on internal (program) alarm setting through the use of the *al\_ ...()* functions.
- Periodic alarms Triggered by timeout condition defined in the alarm setting.

### 9.12.10 Custom page

The Custom page is available from the Version 1.8.3.

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

Custom web page with history graph.

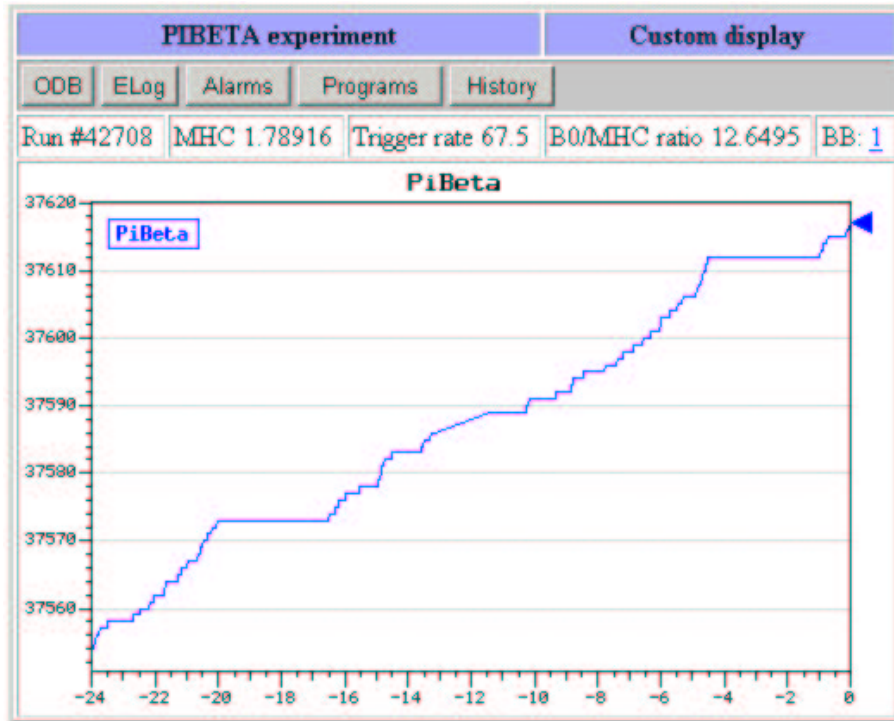


Figure 9.25: Custom web page with history graph.

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

- `<odb src="odb field">` Display ODB field.
- `<odb src="odb field" edit=1>` Display and Editable ODB field.
- `<form method="GET" action="http://hostname.domain:port/CS/<Custom_page_key">` Define method for key access.

- `<meta http-equiv="Refresh" content="60">` Standard page refresh in second.
- `<input type=submit name=cmd value=<Midas_page>>` Define button for accessing Midas web pages. Valid values are the standard midas buttons (Start, Pause, Resume, Stop, ODB, Elog, Alarms, History, Programs, etc).
- `` Reference to an history page.

ODB /Custom/ html field.

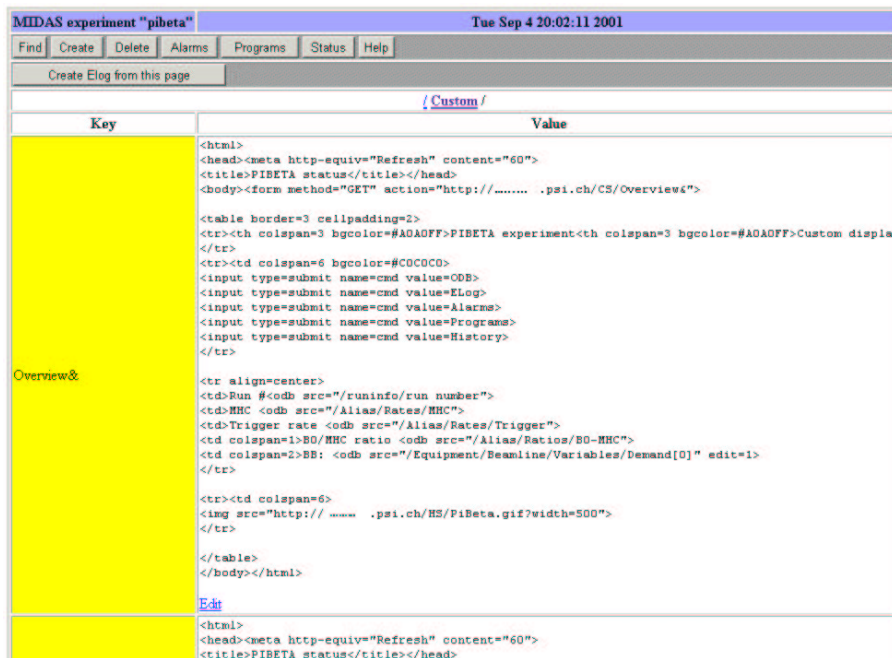


Figure 9.26: ODB /Custom/ html field.

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

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

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

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

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

- The character "&" at the end of the custom key name forces the page to be open within the current frame. If this character is omitted, the page will be spawned into a new frame (default).
- If the custom page name is set to **Status**- (no "&") it will become the default midas Web page!
- html code **Example** mcustom.html

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



```

```


<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.

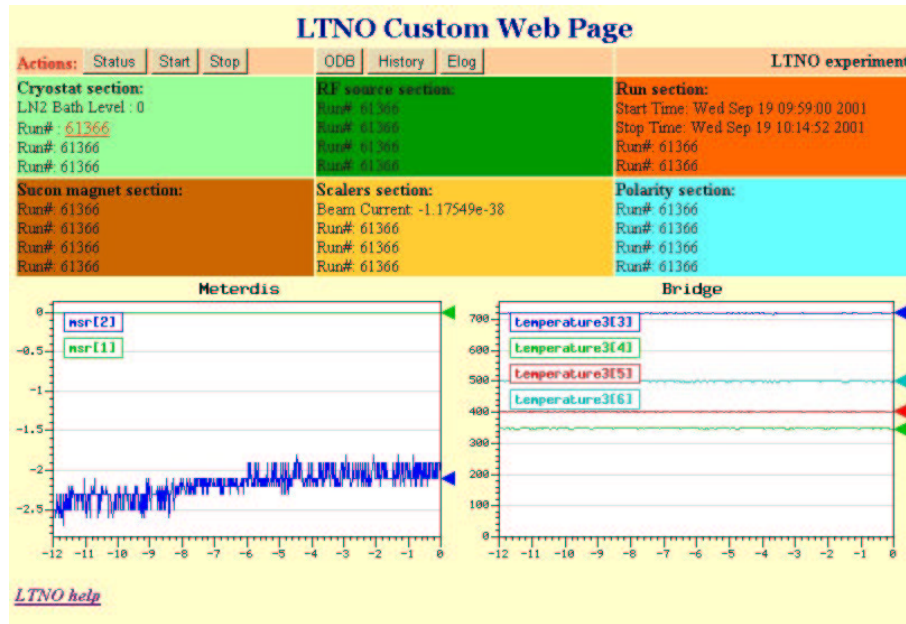


Figure 9.27: web page produced by mcustom.html.

Utilities(p. ??) - Top(p. ??) - Data format(p. ??)



## 9.13 New Documented Features

**Introduction**(p.??) - **Top**(p.??) - **Quick Start**(p.??)

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: 1.9.3-1**
- **Software version: 1.9.3**
- **Latest tarball : 1.9.3-1**
- **Latest RPM : 1.9.2-1**
- **Restructured Midas distribution**
  - In the same effort as the documentation, the midas tree and CVS have been modified. The **download area** now contains separate directories for doc, addon, 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 serve better its purpose. Currently the **MIDAS Analyzer**(p.??) is not complete as well as the **Quick Start**(p.??). This **Doxygen** related files will be made accessible for better update.
- **[Midas Short Course]**
  - During the RealTime Conference 2003 held in Montreal, a short course was offered to introduce the Midas DAQ to the audience. This course (.ppt, .pdf) is now part of the Midas distribution and can be found under the doc/course/ directory as 2 files (part1, part2). The Part 1 describes the basic of the system and its implementation, while part 2 lists specific features. **Part1.pdf**, **Part2.pdf**.
- **[1.9.3]**
  - Support for ROOT files.
  - **mlogger task**(p.??): New Data format **ROOT** and corresponding file extension **root**

- **rmidas task**(p.??): Initial Root/Midas GUI for Histogram and Run control.
- **The ROOT Midas Analyzer**(p.??): New framework for On-line/Offline Root analysis using socket connection.
- **Makefile** for ROOT, remove **MANA\_LITE**, create **HAVE\_ROOT**(p.??), **HAVE\_HBOOK**(p.??).
- New Analyzer **mana**, **hmana**, **rmana** depending on the type of package.

- [1.9.2]

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

- [1.9.1]

- This version addresses several bugs reported in the web interface, history, logger, odbedit and implements new features in particular for the history pages on web interface. The detail list of the modifications can be found in **CHANGELOG** .
  - \* **[EQ\_FRAGMENTED]** Possibility to send extremely large event through the system without modification of the system configuration (see **The Equipment structure**(p.??) )
  - \* **[logger subdir option]** Allows to redirect the data files to a subdirectory based on the time of the creation of the data file (see **ODB /Logger Tree**(p.??) ).
  - \* Option for building an analyzer without the CERN library (HBOOK) (see **Midas build options and operation considerations**(p.??)).
  - \* **[ MOD. REQ.]** This release requires several modifications in the user code in order to compile the 1.9.1.

1. **[db\_get\_value()(p.??) function]** Requires an extra parameter see **Midas Code and Libraries(p.??)**.
2. **[max\_event\_size\_frag]** Required in all the frontend code as follow:

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

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

- **[<1.9.1]**

- Hopefully nobody is still running an older version.

**Introduction(p.??) - Top(p.??) - Quick Start(p.??)**

## 9.14 ODB Structure

### Internal features(p. ??) - Top(p. ??) - Data format(p. ??)

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

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

The ODB information is stored in a "tree/branch" structure where each branch refers to a specific set of data. On the first invocation of the database (first Midas application) a minimal system record will be created. Later on each application will add its own set of parameters to the database depending on its requirement. For instance, starting the ODB for the first time, the tree /**Runinfo**, /**Experiment**, /**System** will be created. The application **mlogger task**(p. ??) 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 awakened by the user in order to provide extra flexibility of the system. Such "dormant" trees are **Alias**, **Script**, **Edit on Start**, **Security**, **Run parameters**.

- ODB /**System Tree**(p. ??)
- ODB /**RunInfo Tree**(p. ??)
- ODB /**Equipment Tree**(p. ??)
- ODB /**Logger Tree**(p. ??)
- ODB /**Experiment Tree**(p. ??)
- ODB /**History Tree**(p. ??)
- ODB /**Alarms Tree**(p. ??)
- ODB /**Script Tree**(p. ??)
- ODB /**Alias Tree**(p. ??)
- ODB /**Elog Tree**(p. ??)
- ODB /**Programs Tree**(p. ??)
- ODB /**Lazy Tree**(p. ??)
- ODB/**EBuilder Tree**(p. ??)
- ODB /**Custom Tree**(p. ??)

## 9.14.1 ODB /System Tree

The system tree contains information specific to each "Midas client" currently 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>

```

### 9.14.2 ODB /RunInfo Tree

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

```

odb -e expt -h host
[host:expt:Running]/>ls -r -l /runinfo

```

Key name	Type	#Val	Size	Last	Opn	Mode	Value
Runinfo	DIR						
State	INT	1	4	2h	0	RWD	3
Online Mode	INT	1	4	2h	0	RWD	1
Run number	INT	1	4	2h	0	RWD	8521
Transition in progress	INT	1	4	2h	0	RWD	0
Requested transition	INT	1	4	2h	0	RWD	0
Start time	STRING	1	32	2h	0	RWD	Thu Mar 23 10:03:44 2000
Start time binary	DWORD	1	4	2h	0	RWD	953834624
Stop time	STRING	1	32	2h	0	RWD	Thu Mar 23 10:03:33 2000
Stop time binary	DWORD	1	4	2h	0	RWD	0

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

- **[Start Time]** Specifies in an ASCII format the time at which the last run has been started.
- **[Start Time binary]** Specifies in a binary format at the time at which the last run has been started This field is useful for time interval computation.
- **[Stop Time]** Specifies in an ASCII format the time at which the last run has been stopped.
- **[Stop Time binary]** Specifies in a binary format the time at which the last run has been stopped. This field is useful for time interval computation.

### 9.14.3 ODB /Equipment Tree

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

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

Example:

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

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

- **Common** : Contains the system information. Should not be overwritten by the user.

- Variables : Contains the equipment data if enabled (see below).
- Settings : Contains the equipment specific information that the user may want to maintain. In the case of a **Slow Control System**(p.??) 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]
                                     [1]
                                     [2]
                                     [3]
                                     [4]
                                     [5]
    RATE                              FLOAT     6     4    1s  0   RWD
                                     [0]
                                     [1]
                                     [2]
                                     [3]
                                     [4]
                                     [5]
  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
```

#### 9.14.4 ODB /Logger Tree

The /Logger ODB tree contains all the relevant information for the Midas logger utility (**mlogger task**(p.??) ) to run properly. This utility provides the mean of storing the physical data retrieved by the frontend to a storage media. The



user has no code to write in order for the system to operate correctly. Its general behavior can be customized and multiple logging channels can be defined. The application supports so far three type of storage devices i.e.: Disk, Tape and FTP channel.

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

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

Data dir Specifies in which directory files produced by the logger should be written.

Once the Logger is running, this **Data\_Dir**(p.??) will be pointing to the location of the midas.log , ODB dump files, history files, message files. In the case of multiple logging channels, the data path for all the channels is defaulted to the same location. In the case where specific directory has to be assigned to each individual logging channel, the field **/logger/channel/<x>/Settings/Filename** can contain the full path of the location of the .mid, .ybs, .asc file. By finding the OS specific **SEPARATOR\_DIR** ("/", "\"). The field **Filename** will overwrite the global **Data\_Dir**(p.??) setting for that particular channel.

History Dir This field is optional and doesn't appear by default in the logger. If present

the location of the **History system**(p.??) files is reassigned to the defined path instead of the default **Data\_Dir**(p.??) .

**Elog Dir** This field is optional and doesn't appear by default in the logger. If present the location of the **Electronic Logbook**(p.??) files is reassigned to the defined path instead of the default **Data\_Dir**(p.??).

**Message file** Specifies the file name for the log file which contains all messages from the MIDAS message system. The message log file is a simple ASCII file, which can be viewed at any time to see a history of what happened in an experiment.

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

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

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

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

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

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

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

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

**active** turns a channel on (1) or off (0). Data is only logged to channels that are active.

**Type** Specify the type of media on which the logging should take place. It can be Disk, Tape or FTP to write directly to a remote computer via FTP.

**Filename** Specify the name of a file in case of a disk logging, where 05d is replaced by the current run number the same way as for the ODB dump files. In the case of a tape logging, the filename specifies a tape device like /dev/nrmt0 or /dev/nst0 under UNIX or \\.\tape0 under Windows NT.

- In FTP mode, the filename specifies the access information for the FTP server. It has the following format

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

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

```
myhost.my.domain,21,john,password,/usr/users/data,run%05d.mid
```

**Format** Specifies the format to be used for writing the data to the logging channel. It can one of the five value: MIDAS, YBOS, ROOT, ASCII and DUMP. The MIDAS and YBOS binary formats **Midas format**(p. ??) and **YBOS format**(p. ??) , respectively. The extension for the file name has to match one of the following.

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

[ODB Dump] Specifies the complete dump of the ODB to the logging channel before and after every run. The ODB content is dumped in one long ASCII string reflecting the status at begin-of-run event and at end-of-run event. These special events have an ID of EVENT\_ID\_BOR and EVENTID\_EOR and a serial number equals to the current run number. An analyzer in the off-line analysis stage can restore the ODB to its online state.

**Log messages** This is a bit-field for logging system messages. If a bit in this field is set, the according system message is written to the logging channel as a

message event with an ID of `EVENT_ID_MESSAGE` (0x8002). The bits are 1 for error, 2 for info, 4 for debug, 8 for user, 16 for log, 32 for talk, 64 for call messages and 255 to log all messages. For an explanation of these messages refer to **Buffer Manager**(p.??), Event ID and Trigger .

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

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

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

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

The **Current** filename will reflect the full path of the saved data file.

### 9.14.5 ODB /Experiment Tree

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

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	#0D&%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@#0%SSDF2

Name Specifies the name of the experiment.

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

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

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

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

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

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

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

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

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

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

**Lock when running** Specifies a fix directory for defining logical link keys to be set in Read only access mode while the run is in progress. The lock when running can

contains logical link to key(s) for setting these keys protection to "read only" while running. In the example below, all the parameters under the declared tree will be switched to read only preventing any parameters modification during the run.

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

**Security** Specifies a fix directory name where information regarding security can be setup. By default, there is no restriction for user to connect locally or remotely to a given experiment. If an access restriction has to be setup in order to protect the experiment from unwilling access, a password mechanism has to be defined.

**Password** Specifies the encrypted password for accessing current experiment.

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

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

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

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

- Password
- Allowed hosts
- Allowed programs
- Web Password

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

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

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

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

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

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

### 9.14.6 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(p.??)** Midas web server when the button "History" on the main status page is pressed.

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



			[1]		1		
Timescale	INT	1	4	36h	0	RWD	3600
Zero ylow	BOOL	1	4	36h	0	RWD	y

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

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

### 9.14.7 ODB /Alarms Tree

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

```
odb -e expt -h host
[host:expt:Stopped]/Alarms>ls -lr
```

Key name	Type	#Val	Size	Last	Opn	Mode	Value
-----							
Alarms	DIR						
Alarm system active	BOOL	1	4	6h	0	RWD	n
Alarms	DIR						
Test	DIR						
Active	BOOL	1	4	31h	0	RWD	n
Triggered	INT	1	4	31h	0	RWD	0
Type	INT	1	4	31h	0	RWD	3
Check interval	INT	1	4	31h	0	RWD	60
Checked last	DWORD	1	4	31h	0	RWD	0
Time triggered first	STRING	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 first	STRING	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] < 900
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 first	STRING	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] < 3000
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 message	BOOL	1	4	31h	0	RWD	y
Write Elog message	BOOL	1	4	31h	0	RWD	n
System message inter	INT	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 message	BOOL	1	4	31h	0	RWD	y
Write Elog message	BOOL	1	4	31h	0	RWD	n
System message inter	INT	1	4	31h	0	RWD	60
System message last	DWORD	1	4	31h	0	RWD	0
Execute command	STRING	1	256	31h	0	RWD	
Execute interval	INT	1	4	31h	0	RWD	0
Execute last	DWORD	1	4	31h	0	RWD	0
Stop run	BOOL	1	4	31h	0	RWD	n

Alarm system active Overall Alarm enable flag.

Alarms Sub-tree defining each individual alarm condition.

Classes Sub-tree defining each individual action to be performed by a pre-defined and requested alarm.

### 9.14.8 ODB /Script Tree

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

```
[host::expt:Stopped]/Script>ls
BNMR Hold
Continue
Real
Test
Kill
```

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

### 9.14.9 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
```

### 9.14.10 ODB /Elog Tree

This branch describes the Elog settings used through the Midas web server. See **mhttpd task(p.??)** 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]
```



Email <...> Email address to where the message should be sent when composing it under "Systems" of the type <...> ,

### 9.14.11 ODB /Programs Tree

System created tree containing task specific characteristics such as the watchdog and alarm condition. See **Alarm System**(p.??) .

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

### 9.14.12 ODB /Lazy Tree

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

Key name	Type	#Val	Size	Last	Opn	Mode	Value
-----							
Lazy	DIR						
Tape	DIR						
Settings	DIR						
Maintain free space	INT	1	4	23h	0	RWD	15
Stay behind	INT	1	4	23h	0	RWD	-1
Alarm Class	STRING	1	32	23h	0	RWD	
Running condition	STRING	1	128	23h	0	RWD	ALWAYS
Data dir	STRING	1	256	23h	0	RWD	/data_onl/current
Data format	STRING	1	8	23h	0	RWD	YBOS
Filename format	STRING	1	128	23h	0	RWD	run%05d.ybs
Backup type	STRING	1	8	23h	0	RWD	Tape
Execute after rewind	STRING	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 writi	STRING	1	64	23h	0	RWD	lazy_prewrite.csh
Execute after writin	STRING	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 per]	FLOAT	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	

### 9.14.13 ODB/EBuilder Tree

Event Builder tree created by **mevb task(p.??)** . More than 2 fragments can be defined. Once the ODB is setup with the user requirments, restarting the task will activate all the fragment defined.

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

### 9.14.14 ODB /Custom Tree

Web string for custom web page. **Editable ONLY** from your Web browser through **Custom page(p.??)** .

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 CELLPADDING=1 CELLSPACING=1 COLS=3 WIDTH="100%" BGCOLOR="#99FF99" >
<caption><b><font face="Georgia"><font color="#000099"><font size=+2>LTN0
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>



<b><i><font color="#000099"><a href="http://myhost.triumf.ca/ltno/index.html">
<br>
LTN0 help</a></font></i></b>
</body>
</html>

```

### 9.14.15 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

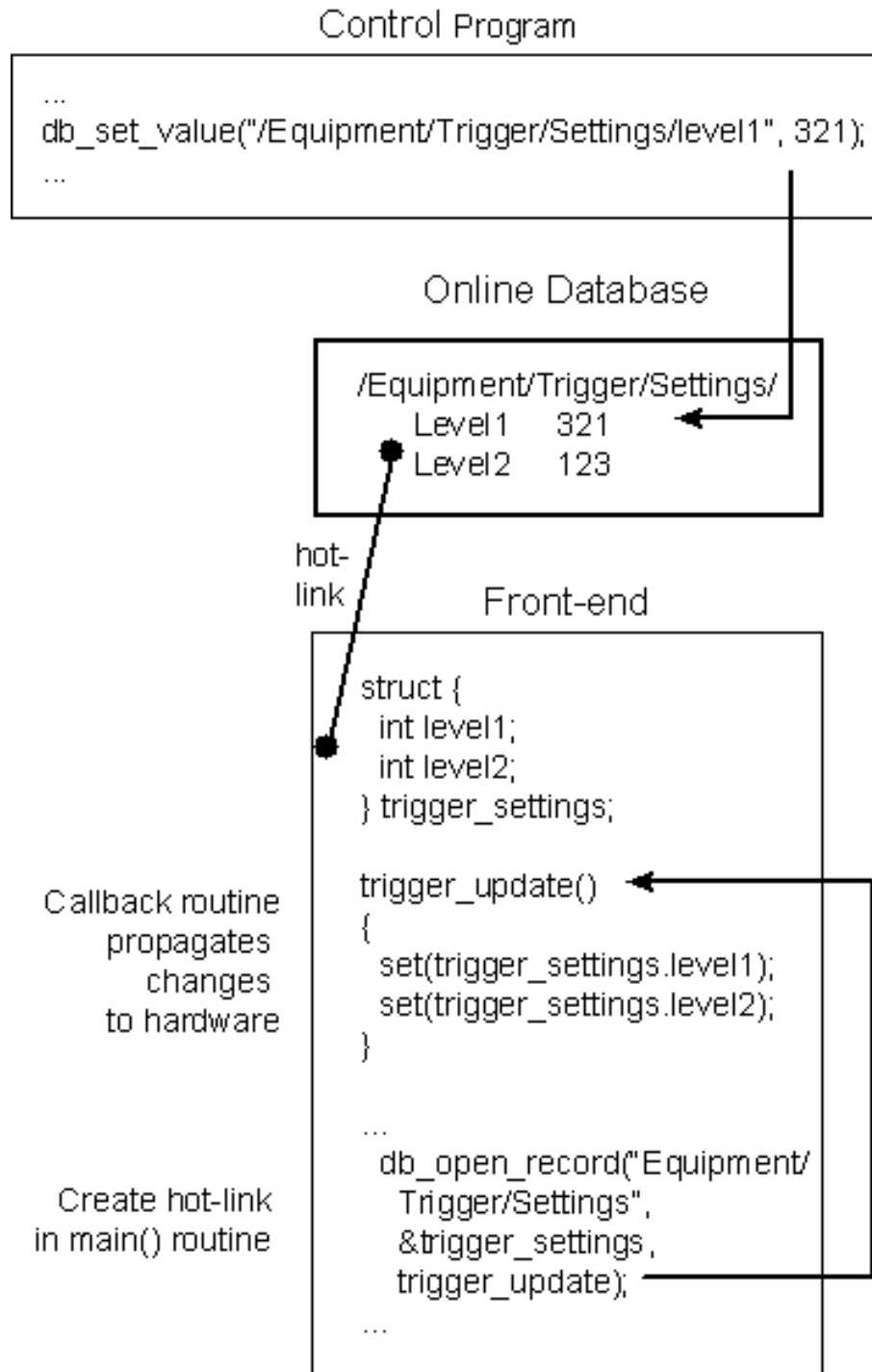


Figure 9.28: HotLink process



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

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

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

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

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

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

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

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

```
#include <experim.h>

TRIGGER_SETTINGS trigger_settings;
```

A hot-link between the ODB values and the C structure is established in the **frontend\_\_init()**(p.??) 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()`(p.??) 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()`(p.??) now establishes the hot-link between the settings in the ODB and the `trigger_settings` structure. Each time the ODB settings are modified, the changes are written to the `trigger_settings` structure and the callback routine `trigger_update()` is executed afterwards. This routine has the task to set the hardware according to the settings in the `trigger_settings` structure.

It may look like:

```

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

```

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

```

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

```

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

```
[local]/>cd /Equipment/Trigger/Settings
[local]Settings>save settings.odb
[local]Settings>set level1 789
[local]Settings>load settings.odb
```

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

```
#include <midas.h>

main()
{
    HNDLE hDB;
    INT level;

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

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

    cm_disconnect_experiment();
}
```

The following figure summarizes the involved components:

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

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

#### 9.14.16 History system

The history system is an add-on capability build in the data logger (see **mlogger task(p.??)** ) to record information in parallel to the data logging. This information is recorded with a time stamp and saved into "data base file" like for later retrieval. One set of file is created per day containing all the requested history events.

The history is working only if the logger is running, but it is not necessary to have any channel enabled.

The definition of the history event is done through two different means:

- **frontend** history event: Each equipment has the capability to generate "history data" if the particular history field value is different then

zero. The value will reflect the periodicity of the history logging (see **The Equipment structure**(p.??)).

- **"Virtual History event"**: Composed within the Online Database under the specific tree "/History/Links" (see **ODB /History Tree**(p.??))

Both definition mode takes effects when the data logger gets a "start run" transition. Any modification during the run is not applied until the next run is started.

frontend history event As mentioned earlier, each equipment can be enabled to generate history event based on the periodicity of the history value (in second!). The content if the event will be completely copied into the history files using the definition of the event as tag names for every element of the event.

The history variable name for each element of the event is composed following one of the rules below:

bank event /**equipment**/**<...>**/Variables/**<bank name>**[] is the only reference of the event, the history name is composed of the bank name followed 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
```

Statistics	DIR	[6]	40
...			

fixed event The names of the individual element under **/equipment/<...>/variables/** will be used for the history name composition.

fixed event with array If the **/equipment/<...>/Settings/Names[]** exists, each element of the array will be referenced using the corresponding name of the **/Settings/Names[]** array.

ODB history event

### 9.14.17 Alarm System

The alarm system is built in and part of the main experiment scheduler. This means no separate task is necessary to benefit 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
<a href="#">Test</a>	Disabled	-	<a href="#">Alarm</a>	/Runinfo/Run number > 100	30327
<a href="#">RF trip</a>	Disabled	-	<a href="#">Pause</a>	/equipment/info odb/variables/RF state = 1	0
<a href="#">Flu monitor</a>	OK	-	<a href="#">Pause</a>	/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 9.29: 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						
<a href="#">/ Programs / Nova 014019 /</a>						
Key				Value		
Auto start				<a href="#">n</a>		
Auto stop				<a href="#">n</a>		
Auto restart				<a href="#">n</a>		
Required				<a href="#">n</a>		
Start command				<a href="#">(empty)</a>		
Alarm Class				<a href="#">(empty)</a>		
Checked last				<a href="#">965499475 (0x398C5A53)</a>		
Alarm count				<a href="#">0 (0x0)</a>		
Watchdog timeout				<a href="#">10000 (0x2710)</a>		

Figure 9.30: 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	
<a href="#">ODBEdit</a>	midtis01	-	No	<a href="#">Stop ODBEdit</a>
<a href="#">TRINAT_FE</a>	codaq01	-	No	<a href="#">Stop TRINAT_FE</a>
<a href="#">MStatus</a>	midtis01	-	No	<a href="#">Stop MStatus</a>
<a href="#">Logger</a>	midtis01	-	No	<a href="#">Stop Logger</a>
<a href="#">Nova_014019</a>	midtis01	-	No	<a href="#">Stop Nova_014019</a>

Figure 9.31: Midas Web Alarm Program status display

Internal features(p.??) - Top(p.??) - Data format(p.??)

## 9.15 Quick Start

**Components**(p. ??) - **Top**(p. ??) - **Internal features**(p. ??)

This section is under revision to better reflect the latest installation and basic operation of the Midas package.

... This section will... describes step-by-step the installation procedure of the Midas package on several platform as well as the procedure to run a demo sample experiment. In a second stage, the frontend or the analyzer can be moved to another computer to test the remote connection capability.

The Midas Package source and binaries can be found at : PSI or at TRIUMF . An online 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**(p. ??) and **Windows installation**(p. ??).

### 9.15.1 Linux installation

#### 1. Extraction:

- **Compressed files** The compressed file contains the source and binary code. It does expand under the directory name of **midas**. This extraction can be done at the user level.

```
cd /home/mydir
tar -zxvf midas-1.9.x.tar.gz
```

The midas directory structure will be composed of several subdirectories such as:

```
>ls
COPYING  doc/      examples/ include/  linux/    makefile.nt  mscb/  utils/
CVS/     drivers/  gui/      java/    Makefile* mcleanup*  src/   vxworks/
```

- **RPM** Current RPM is not fully up-to-date. We suggest that you use the compressed files or the 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 be used for respectively **checking out** (first time) and **updating** the full midas tree.

```
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 files in the **/usr/local/** directories. This operation requires superuser privilege. The open "install" from the Makefile will automatically do this installation for you.



```
cd /home/mydir/midas
su -
make install
```

3. **Configuration:** Several system files needs to be modified for the full Midas implementation.

- **/etc/services :** For remote access. Inclusion of the midas service. Add following line:

```
# midas service
midas      1175/tcp      # Midas server
```

- **/etc/xinetd.d/midas :** Daemon definition. Create new file named **midas**

```
service midas
{
    flags                = REUSE NOLIBWRAP
    socket_type          = stream
    wait                 = no
    user                 = root
    server                = /usr/local/bin/mserver
    log_on_success        += USERID HOST PID
    log_on_failure        += USERID HOST PID
    disable               = no
}
```

- **/etc/ld.so.conf :** Dynamic Linked library list. Add directory pointing to location of the midas.so file (add /usr/local/lib).

```
/usr/local/lib
```

The system environment **LD\_LIBRARY\_PATH** can used instead.

- **/etc/exptab :** Midas Experiment definition file (see below).

4. **Experiment definition:** Midas system supports multiple experiment running contemporary on the same computer. Even though it may not be efficient, this capability makes sense when the experiments are simple detector lab setups which shared hardware resources for data collection. In order to support this feature, Midas requires a uniquely identified set of parameter for each experiment that is used to define the location of the Online Database.

Every experiment under Midas has its own ODB. In order to differentiate them, an experiment name and directory are assigned to each experiment. This allows several experiments to run concurrently on the same host using a common Midas installation.

Whenever a program participating in an experiment is started, the experiment name can be specified as a command line argument or as an environment variable.

A list of all possible running experiments on a given machine is kept in the file **exptab**. This file **exptab** is expected by default to be located under **/etc/exptab**. This can be overwritten by the **Environment variables(p.??) MIDAS\_EXPTAB(p.??)**.

**exptab** file is composed of one line per experiment definition. Each line contains three parameters, i.e: **experiment name experiment directory name user name**. Example:

```
#
# Midas experiment list
midas /home/midas/online midas
decay /home/slave/decay_daq slave
```

Experiments not defined into **exptab** are not accessible remotely, but can still be accessed locally using the **Environment variables(p.??) MIDAS\_DIR(p.??)** if defined. This environment superceed the **exptab** definition.

5. **Compilation & Build:** You should be able to rebuild the full package once the Midas tree structure has been placed in your temporary directory. The compilation and link will try to generate the **rmidas** application which requires **ROOT**. The application **mana** will also be compiled for **HBOOK** and **ROOT**. Look in the make listing below for the **HAVE\_HBOOK(p.??)**, **HAVE\_ROOT(p.??)**.

```
> cd /home/mydir/midas
> make
cc -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/lib/midas.o src/midas.c
cc -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/lib/system.o src/system.c
cc -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/lib/mrpc.o src/mrpc.c
cc -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/lib/odb.o src/odb.c
cc -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/lib/ybos.o src/ybos.c
cc -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/lib/ftplib.o src/ftplib.c
rm -f linux/lib/libmidas.a
ar -crv linux/lib/libmidas.a linux/lib/midas.o linux/lib/system.o linux/lib/mrpc.o
linux/lib/odb.o linux/lib/ybos.o linux/lib/ftplib.o
a - linux/lib/midas.o
a - linux/lib/system.o
a - linux/lib/mrpc.o
a - linux/lib/odb.o
a - linux/lib/ybos.o
a - linux/lib/ftplib.o
rm -f linux/lib/libmidas.so
ld -shared -o linux/lib/libmidas.so linux/lib/midas.o linux/lib/system.o
linux/lib/mrpc.o linux/lib/odb.o linux/lib/ybos.o linux/lib/ftplib.o -lutil
-lpthread -lc
```

```

cc -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/lib/mana.o src/mana.c
cc -Dextname -DHAVE_HBOOK -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib
-DINCLUDE_FTPLIB -DOS_LINUX -fPIC -o linux/lib/hmana.o src/mana.c
...
g++ -DHAVE_ROOT -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB
-DOS_LINUX -fPIC -D_REENTRANT -I/home1/midas/ root/include -o linux/lib/rmana.o
src/mana.c
g++ -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINU
-fPIC -o linux/lib/mfe.o src/mfe.c
cc -Dextname -c -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib
-DINCLUDE_FTPLIB -DOS_LINUX -fPIC -o linux/lib/fal.o src/fal.c
...
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mserver src/mserver.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mhttpd src/mhttpd.c src/mgd.c -lmidas -lutil -lpthread -lm
g++ -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-DHAVE_ROOT -D_REENTRANT -I/home1/midas/root/include
-o linux/bin/mlogger src/mlogger.c -lmidas
-L/home1/midas/root/lib -lCore -lCint -lHist -lGraf -lGraf3d -lGpad -lTree
-lRint -lPostscript -lMatrix -lPhysics -lpthread -lm -ldl -rdynamic -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/odbedit src/odbedit.c src/cmdedit.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mtape utils/mtape.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mhist utils/mhist.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mstat utils/mstat.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mcnaf utils/mcnaf.c drivers/bus/camacrpc.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mdump utils/mdump.c -lmidas -lz -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/lazylogger src/lazylogger.c -lmidas -lz -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mchart utils/mchart.c -lmidas -lutil -lpthread
cp -f utils/stripchart.tcl linux/bin/.
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/webpaw utils/webpaw.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/odbhist utils/odbhist.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/melog utils/melog.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/mlxspeaker utils/mlxspeaker.c -lmidas -lutil -lpthread
cc -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-o linux/bin/dio utils/dio.c -lmidas -lutil -lpthread
g++ -g -O2 -Wall -Iinclude -Idrivers -Llinux/lib -DINCLUDE_FTPLIB -DOS_LINUX -fPIC
-DHAVE_ROOT -D_REENTRANT -I/home1/midas/root/include -o linux/bin/rmidas src/rmidas.c
-lmidas -L/home1/midas/root/lib -lCore -lCint -lHist -lGraf -lGraf3d -lGpad
-lTree -lRint -lPostscript -lMatrix -lPhysics -lGui -lpthread -lm -ldl -rdynamic
-lutil -lpthread

```

6. **Demo examples:** The midas file structure contains examples of code which can be (should be) used for template. In the **mi-**

`das/examples/experiment` you will find a full set of frontend and analysis code.

```
> 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
-lns1 -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 -lpthread -lm -ldl
-rdynamic -lThread -lm -lz -lutil -lns1 -lpthread
>
> 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
```

### 9.15.2 Windows installation

1. Extraction:

2. Installation:

3. Configuration:

4. Experiment definition:

5. Compilation:

6. Demo examples:

Components(p. ??) - Top(p. ??) - Internal features(p. ??)

## 9.16 Utilities

### Internal features(p. ??) - Top(p. ??) - Data format(p. ??)

The Midas system provides several off-the-shelf programs to control, monitor, debug the data acquisition system. Starting with the main utility (odbedit) which provide access to the Online data base and run control.

- **odbedit task(p. ??)** : Online Database Editor
  - **ODB Structure(p. ??)**
- **mstat task(p. ??)** : Midas ASCII status report
- **analyzer task(p. ??)** : Midas data analyzer
  - **MIDAS Analyzer(p. ??)**
- **mlogger task(p. ??)** : Midas data logger
- **lazylogger task(p. ??)** : Background data logger
- **mdump task(p. ??)** : Event dump application
- **mevb task(p. ??)** : Event Builder application
- **mspeaker, mlxspeaker tasks(p. ??)** : Speech synthesizer
- **mcnaf task(p. ??)** : CAMAC standalone application
- **mhttpd task(p. ??)** : Midas Web server
- **Elog task(p. ??)** : Electronic entry application
- **mhist task(p. ??)** : History retrieval application
- **mchart task(p. ??)** : Standalone Chart display application
- **mtape task(p. ??)** : Tape device manipulator
- **dio task(p. ??)** : Direct IO provider
- **stripchart.tcl file(p. ??)** : Tcl/Tk for chart display
- **rmidas task(p. ??)** : Root/Midas Simple GUI application
- **hvedit task(p. ??)** : High Voltage Slow Control GUI

### 9.16.1 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**(p. ??) for more information.

- **Arguments**

- h : help.
- h hostname : Specifies host to connect to. Must be a valid IP host name. This option supersedes the MIDAS\_SERVER\_HOST environment variable.
- e exptname : Specifies the experiment to connect to. This option supersedes the MIDAS\_EXPT\_NAME environment variable.
- c command : Perform a single command. Can be used to perform operations in script files.
- c @commandFile : Perform commands in sequence found in the commandFile.
- s size : size in byte (for creation). Specify the size of the ODB file to be created when no share file is present in the experiment directory (default 128KB).
- d ODB tree : Specify the initial entry ODB path to go to.

- **Usage** ODBedit is the MIDAS run control program. It has a simple command line interface with command line editing similar to the UNIX tcsh shell. Following edit keys are implemented:

- Backspace Erase 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 file system. Most commands are similar to UNIX file commands like ls, cd, chmod, ln etc. The help command displays a short description of all commands.

```
[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 memory <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
```

## 9.16.2 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(p.??))
-e exptname : experiment name (see odbedit task(p.??))
-l : loop. Forces mstat to remain in a display loop. Enter "!" to
    terminate the command.
-w time : refresh rate in second. Specifies the delay in second before refreshing
    the screen with up to date information. Default: 5 seconds. Has to
    be used in conjunction with -l switch. Enter "R" to refresh screen on
    next update.
```

### • Usage

```

>mstat -l
*-v1.8.0- MIDAS status page -----Mon Apr  3 11:52:52 2000-*
Experiment:chaos      Run#:8699      State:Running      Run time :00:11:34
Start time:Mon Apr  3 11:41:18 2000

FE Equip.   Node           Event Taken   Event Rate[/s]  Data Rate[Kb/s]
B12Y        pcch02             67            0.0             0.0
CUM_Scaler  vwchaos           23            0.2             0.2
CHV          pcch02            68            0.0             0.0
KOS_Scalers vwchaos          330            0.4             0.6
KOS_Trigger vwchaos        434226         652.4           408.3
KOS_File     vwchaos            0            0.0             0.0
Target      pcch02            66            0.0             0.0

Logger Data dir: /scr0/spring2000      Message File: midas.log
Chan.   Active Type   Filename      Events Taken   KBytes Taken
  0      Yes   Disk   run08699.ybs   434206         4.24e+06

Lazy Label   Progress  File name      #files      Total
cni-53       100[%]    run08696.ybs   15           44.3[%]

Clients:  MStatus/koslx0      Logger/koslx0      Lazy_Tape/koslx0
          CHV/pcch02      MChart1/umelba     ODBEdit/koslx0
          CHAOS/vwchaos    ecl/koslx0         Speaker/koslx0
          MChart/umelba    targetFE/pcch02    HV_MONITOR/umelba
          SUSIYBOS/koslx0  History/kosal2     MStatus1/dasdevpc
*-----*

```

### 9.16.3 analyzer task

**analyzer** is the main online / offline event analysis application. **analyzer** uses fully the **ODB** capabilities as all the analyzer parameters are dynamically controllable from the Online Database editor **odbedit task**(p.??).

For more detailed information see **MIDAS Analyzer**(p.??)

- Arguments

- h : help
- h hostname : host name (see **odbedit task**(p.??))
- e exptname : experiment name (see **odbedit task**(p.??))
- D : start program as a daemon (UNIX only).
- i \<filename1> <filename2> : Input file name. May contain a '%05d' to be replaced by the run number. Up to ten input files can be specified in one "-i" statement.
- o <filename> : Output file name. Extension may be .mid (MIDAS binary), .asc (ASCII) or .rz (HBOOK). If the name contains a '%05d', one output file is generated for each run.
- r <range> : Range of run numbers to analyzer like "-r 120 125" to analyze runs 120 to 125 (inclusive). The "-r" flag must be used with a '%05d' in the input file name.

- n <count> : Analyze only "count" events.
- n <first> <last> : Analyze only events from "first" to "last".
- n <first> <last> <n> : Analyze every n-th event from "first" to "last".
  - f : Filter events. Write original events to output file only if analyzer accepts them (doesn't return ANA\_SKIP).
- c <filename1> <filename2> : Configuration file name(s). May contain a '%05d' to be replaced by the run number. Up to ten files can be specified in one "-c" statement.
- p <param=value> : Set individual parameters to a specific value. Overrides any setting in configuration files
  - w : Produce row-wise N-tuples in output .rz file. By default, column-wise N-tuples are used.
  - v : Verbose output.
  - d : Debug flag when started the analyzer from a debugger. Prevents the system to kill the analyzer when the debugger stops at a breakpoint
  - q : Quiet flag. If set, don't display run progress in offline mode.
  - l : If set, don't load histos from last.rz when running online.
  - L : HBOOK LREC size. Default is 8190.
- P <ODB tree> Protect an ODB subtree from being overwritten with the online data when ODB gets loaded from .mid.

#### • 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
>analyzer -i run00023.mid -o run00023.rz -w
>analyzer -i run%05d.mid -o runall.rz -r 23 75 -w
```

### 9.16.4 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 flags is enabled in the frontend equipment (see **The Equipment structure**(p.??) or if the ODB tree /History/Links is defined (See **History system**(p.??)). See the **ODB /Logger Tree**(p.??) for reference on the tree structure.

- **Arguments**

- h : help
- h hostname : host name (see **odbedit task(p.??)**)
- e exptname : experiment name (see **odbedit task(p.??)**)
- D : start program as a daemon (UNIX only).

- **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 defined while running but effect will be taken place only at the following begin of run transition.
- The default setting defines a data "Midas" format with a file name of the type "run\%05d.mid". Make sure this is the requested setting for your experiment.
- Once the mlogger is running, you should be able to monitor its state through the **mstat task(p.??)** or through a web browser if the **mhttpd task(p.??)** is running.

### 9.16.5 lazylogger task

**lazylogger** is an application which decouples the data acquisition 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, retention, reposition imply that the data acquisition has to hold until operation completion. By using **mlogger** to log data to disk in a first stage and then using **lazylogger** to copy or move the stored files to the "slow device" we can keep the acquisition running without interruption.

- Multiple 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 successfully manipulate by the Lazylogger channel. This list won't exists until the first 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 defined lazylogger channels.

- **ODB parameters (Settings/)**

Settings	DIR								
Maintain free space(%)	INT	1	4	3m	0	RWD	0		
Stay behind	INT	1	4	3m	0	RWD	-3		
Alarm Class	STRING	1	32	3m	0	RWD			
Running condition	STRING	1	128	3m	0	RWD	ALWAYS		
Data dir	STRING	1	256	3m	0	RWD	/home/midas/online		
Data format	STRING	1	8	3m	0	RWD	MIDAS		
Filename format	STRING	1	128	3m	0	RWD	run%05d.mid		
Backup type	STRING	1	8	3m	0	RWD	Tape		
Execute after rewind	STRING	1	64	3m	0	RWD			
Path	STRING	1	128	3m	0	RWD			
Capacity (Bytes)	FLOAT	1	4	3m	0	RWD	5e+09		
List label	STRING	1	128	3m	0	RWD			
Execute before writing file	STRING	1	64	11h	0	RWD	lazy_prewrite.csh		
Execute after writing file	STRING	1	64	11h	0	RWD	rundb_addrun.pl		
Modulo.Position	STRING	1	8	11h	0	RWD	2.1		
Tape Data Append	BOOL	1	4	11h	0	RWD	y		

- **[Maintain free space]** As the Data Logger (mlogger) runs independently from the Lazylogger, the disk will contains all the recorded data files. Under this condition, Lazylogger can be instructed to "purge" the data logging device (disk) after successful backup of the data onto the "slow device". The *Maintain* free space(%) parameter controls (if none zero) the percentage of disk space required to be maintain free.

\* The condition for removing a data file is defined as:

The data file corresponding to the given run number following the format declared under "Settings/Filename format" IS PRESENT on the "Settings/Data Dir" path. AND The given run number appears anywhere under the "List/" directory of ALL the Lazy channel having the same "Settings/Filename format" as this channel. AND The given run number appears anywhere under the "List/" directory of that channel

- [Stay behind] This parameter defines how many consecutive data file 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 file untouched 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 file sitting on the disk at the "Data Dir" path- for backup. If the "Stay behind" is positive, lazylogger will **backup** starting from- x behind the current acquisition run number. Run 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 activated. 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 satisfied such as "Low beam intensity". In this later case, condition based on a single field of the ODB can be given to establish when the application should be active.

\* **Example :**

```
odbedit> set "Running condition" WHILE_ACQ_NOT_RUNNING
odbedit> set "Running condition" "/alias/max_rate < 200"
```

- [Data dir] Specify the Data directory path of the data files. By default if the "/Logger/Data Dir" is present, the pointed value is taken otherwise the current directory where lazylogger has been started is used.
- [Data format] Specify the Data format of the data files. Currently supported format are: **MIDAS** or **YBOS**.

- **[Filename format]** Specify the file format of the data files. Same format as given for the data logger.
- **[Backup type]** Specify the "slow device" backup type. Default **Tape**. Can be **Disk** or **Ftp**.
- **[Execute after rewind]** Specify a script to 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" field to prevent further backup (see below). It will also rewind the stream device if possible.
- **[List label]** Specify a label for a set of backed up files to the "slow device". This label is used only internally 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 field and therefore prevent any further backup until a none empty label list is entered again. In the other hand the list label will remain under the **/List** key to display all run being backed up until the corresponding files have been removed from the disk.
- **[Exec preW file]** Permits to run a script before the beginning of the lazy job. The **Arguments** passed to the scripts are: input file name , output file name, current block number.
- **[Exec postW file]** Permits to run a script after the completion of the lazy job. The **Arguments** passed to the scripts are: list label, current job number, source path, file name, file size in MB, current block number.
- **[Modulo.Position]** This field is for multiple 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 field assignment should be :

Channel	Field	Run#
Lazy_1	3.0	21, 24, 27, ...
Lazy_2	3.1	22, 25, 28, ...
Lazy_3	3.2	23, 26, 29, ...

- **[Tape Data Append]** Enable the spooling of the Tape device to the **End\_of\_Device** (EOD) before starting the lazy job. This command

is valid only for "Backup Type" Tape. If this flag is not 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 file 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 field to match your requirement.

```
> odbedit -e midas
[local:midas:Stopped]/>cd /Lazy/tape/
[local:midas:Stopped]tape>ls
[local:midas:Stopped]tape>ls -lr
```

Key name	Type	#Val	Size	Last	Opn	Mode	Value
tape	DIR						
Settings	DIR						
Maintain free space(%)	INT	1	4	3m	0	RWD	0
Stay behind	INT	1	4	3m	0	RWD	-3
Alarm Class	STRING	1	32	3m	0	RWD	
Running condition	STRING	1	128	3m	0	RWD	ALWAYS
Data dir	STRING	1	256	3m	0	RWD	/home/midas/online
Data format	STRING	1	8	3m	0	RWD	MIDAS
Filename format	STRING	1	128	3m	0	RWD	run%05d.mid
Backup type	STRING	1	8	3m	0	RWD	Tape
Execute after rewind	STRING	1	64	3m	0	RWD	
Path	STRING	1	128	3m	0	RWD	
Capacity (Bytes)	FLOAT	1	4	3m	0	RWD	5e+09
List label	STRING	1	128	3m	0	RWD	
Statistics	DIR						
Backup file	STRING	1	128	3m	0	RWD	none
File size [Bytes]	FLOAT	1	4	3m	0	RWD	0
KBytes copied	FLOAT	1	4	3m	0	RWD	0
Total Bytes copied	FLOAT	1	4	3m	0	RWD	0
Copy progress [%]	FLOAT	1	4	3m	0	RWD	0
Copy Rate [bytes per s]	FLOAT	1	4	3m	0	RWD	0
Backup status [%]	FLOAT	1	4	3m	0	RWD	0
Number of Files	INT	1	4	3m	0	RWD	0
Current Lazy run	INT	1	4	3m	0	RWD	0

```
[local:midas:Stopped]tape>cd Settings/
[local:midas:Stopped]Settings>set "Data dir" /data
[local:midas:Stopped]Settings>set "Capacity (Bytes)" 15e9
```

- **[Step 3]** Start lazylogger in the background



```
>lazylogger -c Tape -D
```

- **[Step 4]** At this point the lazylogger is running and waiting for the "list label" to be defined before starting the copy procedure. **mstat task(p.??)** 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 REMOVED
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 864.020MB file NEW
Fri Mar 24 15:07:35 2000 [Lazy_Tape] 8352 (rm:25854ms) /scr0/spring2000/run08352.ybs file REMOVED
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 REMOVED
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 857.677MB file NEW
Fri Mar 24 15:50:15 2000 [Lazy_Tape] 8354 (rm:28867ms) /scr0/spring2000/run08354.ybs file REMOVED
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(p. ??)** can be used for ODB as well as lazylogger recovery.

### 9.16.6 mdump task

This application allows to "peep" into the data flow in order to display a snapshot 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 containing 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 recommended 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 diagnostic.
- w time : Insert wait in [sec] between each display.

-x filename : Input channel. data file name of data device. (def:online)  
 -y : Display consistency check only.  
 -z buffer name : Midas buffer name to attach to (def:SYSTEM)

- Additional **Arguments** for Offline

-x -h : help for offline use.  
 -t type : Bank format (Midas/Ybos). >>> if -x is a /dev/xxx, -t has to be specified.  
 -r # : skip record(YBOS) or event(MIDAS) to #.  
 -w what : Header, Record, Length, Event, Jbank\_list (def:E) >>> Header & Record are not supported for MIDAS as it has no physical record structure.

- **Usage** mdump can operate on either data stream (online) or on save-set data file. Specific help is available for each mode.

```
> mdump -h
> mdump -x -h
```

```
Tue> mdump -x run37496.mid | more
----- Event# 0 -----
----- Event# 1 -----
Evid:0001- Mask:0100- Serial:1- Time:0x393c299a- Dsize:72/0x48
#banks:2 - Bank list:-SCLRRATE-

Bank:SCLR Length: 24(I*1)/6(I*4)/6(Type) Type:Integer*4
1-> 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000

Bank:RATE Length: 24(I*1)/6(I*4)/6(Type) Type:Real*4 (FMT machine dependent)
1-> 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
----- Event# 2 -----
Evid:0001- Mask:0004- Serial:1- Time:0x393c299a- Dsize:56/0x38
#banks:2 - Bank list:-MMESMMOD-

Bank:MMES Length: 24(I*1)/6(I*4)/6(Type) Type:Real*4 (FMT machine dependent)
1-> 0x3de35788 0x3d0b0e29 0x00000000 0x00000000 0x3f800000 0x00000000

Bank:MMOD Length: 4(I*1)/1(I*4)/1(Type) Type:Integer*4
1-> 0x00000001
----- Event# 3 -----
Evid:0001- Mask:0008- Serial:1- Time:0x393c299a- Dsize:48/0x30
#banks:1 - Bank list:-BMES-

Bank:BMES Length: 28(I*1)/7(I*4)/7(Type) Type:Real*4 (FMT machine dependent)
1-> 0x443d7333 0x444cf333 0x44454000 0x4448e000 0x43bca667 0x43ce0000 0x43f98000
----- Event# 4 -----
Evid:0001- Mask:0010- Serial:1- Time:0x393c299a- Dsize:168/0xa8
#banks:1 - Bank list:-CMES-

Bank:CMES Length: 148(I*1)/37(I*4)/37(Type) Type:Real*4 (FMT machine dependent)
```

```

1-> 0x3f2f9fe2 0x3ff77fd6 0x3f173fe6 0x3daeffe2 0x410f83e8 0x40ac07e3 0x3f6ebfd8 0x3c47ffde
9-> 0x3e60ffda 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x3f800000
17-> 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
25-> 0x3f800000 0x3f800000 0x3f800000 0x00000000 0x3f800000 0x00000000 0x3f800000 0x3f800000
33-> 0x3f800000 0x3f800000 0x3f800000 0x3f800000 0x00000000
----- Event# 5 -----
Evid:0001- Mask:0020- Serial:1- Time:0x393c299a- Dsize:32/0x20
#banks:1 - Bank list:-METR-

Bank:METR Length: 12(I*1)/3(I*4)/3(Type) Type:Real*4 (FMT machine dependent)
1-> 0x00000000 0x39005d87 0x00000000
...

```

- **Example**

```
> mdump -j
```

### 9.16.7 mevb task

**mevb** is a event builder tool assembling multiple event fragment received from different event buffers.

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 synchronization of the fragment collection is left to the user which is done usually through specific hardware mechanism. Once the fragments are composed in each frontend, they are sent to the "Event Builder" (eb) where the serial number (pheader->serial\_number) of each fragment is compared one event at a time for serial match. In case of match, a new event will be composed with its own event ID and serial number followed by all the expected fragments. The composed event is then sent to the next stage which is usually the data logger (mlogger).

The **mhttpd task(p.??)** will present a specific section on its page for the **mevb task(p.??)** if running.

- **Arguments**

```

-h : help
-h hostname : host name
-e exptname : experiment 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**(p. ??) for more details

### 9.16.8 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
```

### 9.16.9 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**(p. ??)).
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**(p. ??)) from the **Building Options**(p. ??) has included the **HAVE\_CAMAC**(p. ??) 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.mcna` will build a collection of `mcna` applications which are hardware dependent, see **Example** below:

- **[miocna]** cna application using the declared CAMAC hardware DRIVER (kcs2927 in this case). To be used with **dio-** CAMAC application starter (see **dio task(p.??)**).
- **[mwecna]** cna application using the WI-E-N-ER PCI/CAMAC interface (see **CAMAC drivers(p.??)**). Please contact <mailto:midas@triumf.ca> for further information.
- **[mcna]** cna application using the CAMAC RPC capability of any Midas frontend program having CAMAC access.
- **[mdrvcna]** cna application using the Linux CAMAC driver for either kcs2927, kcs2926, dsp004. This application would require to have the proper Linux module loaded in the system first. Please contact <mailto:midas@triumf.ca> for further information.

```
Thu> cd /midas/utils
Thu> make -f makefile.mcna DRIVER=kcs2927
gcc -O3 -I../include -DOS_LINUX -c -o mcna.o mcna.c
gcc -O3 -I../include -DOS_LINUX -c -o kcs2927.o ../drivers/bus/kcs2927.c
gcc -O3 -I../include -DOS_LINUX -o miocna mcna.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 mwecna mcna.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 mcna mcna.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 mdrvna mcna.o camaclx.o ../linux/lib/libmidas.a -lutil
rm *.o
```

- **Running application**

- Direct CAMAC access: This requires the computer to have the proper CAMAC interface installed and the **BASE ADDRESS** matching the value defined in the corresponding CAMAC driver. For kcs2926.c, kcs2927.c, dsp004.c, hyt1331.c, the base address (CAMAC\_BASE) is set to 0x280.

```
>dio miocna
```

- 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**

.....

### 9.16.10 Elog task

Electronic Log utility. Submit full Elog entry to the specified Elog port.

- **Arguments**

- h : help
- h hostname : host name
- l exptname or logbook
- u username password
- f <attachment> : up to 10 files.
  - -a <attribute>=<value> : up to 20 attributes. The attribute "Author=..." must at least be present for submission of Elog.
  - -m <textfile> | text> Arguments with blanks must be enclosed in quotes. The elog message can either be submitted on the command line or in a file with the -m flag. Multiple attributes and attachments can be supplied.

- **Usage** By default the attributes are "Author", "Type", "System" and "Subject". The "Author" attribute has to be present in the elog command in order to successfully submit the message. If multiple attributes are required append before "text" field the full specification of the attribute. In case of multiple attachment, only one "-f" is required followed by up to 10 file names.

```
>elog -h myhost -p 8081 -l myexpt -a author=pierre "Just a elog message"
>elog -h myhost -p 8081 -l myexpt -a author=pierre -f file2attach.txt \
    "Just this message with an attachment"
>elog -h myhost -p 8081 -l myexpt -a author=pierre -m file_containing_the_message.txt
>elog -h myhost -p 8081 -l myexpt -a Author=pierre -a Type=routine -a system=general \
    -a Subject="my test" "A full Elog message"
```

- **Remarks**

### 9.16.11 mhist task

History data retriever.

- **Arguments**

- h : help
- e Event ID : specify event ID
- v Variable Name : specify variable name for given Event ID
- i Index : index of variables which are arrays
- i Index1:Index2 index range of variables which are arrays (max 50)
- t Interval : minimum interval in sec. between two displayed records
- h Hours : display between some hours ago and now
- d Days : display between some days ago and now
- f File : specify history file explicitly
- s Start date : specify start date DDMMYY[.HHMM[SS]]
- p End date : specify end date DDMMYY[.HHMM[SS]]
  - l : list available events and variables
  - b : display time stamp in decimal format
  - z : History directory (def: cwd).

- **Usage**

- **Example**

```

--- All variables of event ID 9 during last hour with at least 5 minutes interval.
> mhist
Available events:
ID 9: Target
ID 5: CHV
ID 6: B12Y
ID 20: System

Select event ID: 9

Available variables:
0: Time
1: Cryostat vacuum
2: Heat Pipe pressure
3: Target pressure
4: Target temperature
5: Shield temperature
6: Diode temperature

Select variable (0..6,-1 for all): -1

```



How many hours: 1

Interval [sec]: 300

Date	Time	Cryostat vacuum	Heat Pipe pressure	Target pressure	Target temperature	Shield temperature
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.

```

mhlist -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(p.??)`).
- **Example**

Midas Web History display.

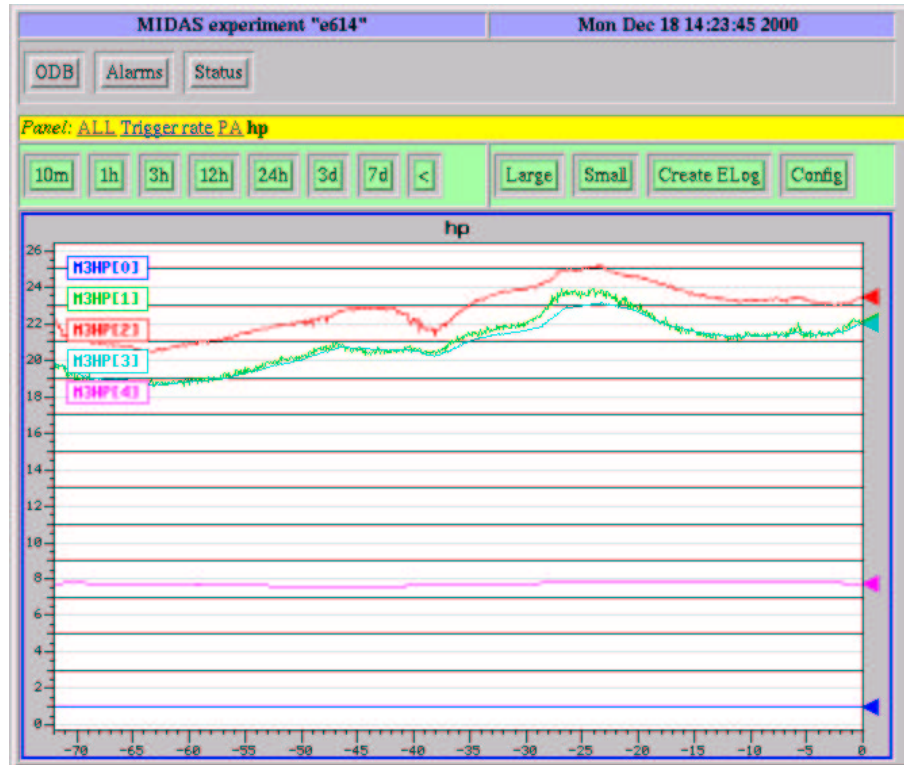


Figure 9.32: Midas Web History display.

### 9.16.12 mchart task

mchart is a periodic data retriever of a specific path in the ODB which can be used in conjunction with a stripchart graphic program.

- In the first of two step procedure, a specific path in the ODB can be scanned for composing a configuration file by extracting all numerical data references **file.conf** .
- In the second step the mchart will produce at fix time interval a refreshed data file containing the values of the numerical data specified in the configuration file. This file is then available for a stripchart program to be used for chart recording type of graph.

Two possible stripchart available are:

- **gstripchart** The configuration file generated by mchart is compatible with the GNU stripchart which permit sophisticated data equation manip-

ulation. In the other hand, the data display is not very fancy and provide just a basic chart recorder.

- **stripchart.tcl file(p. ??)** This tcl/tk application written by Gertjan Hofman provides a far better graphical chart recorder display tool, it also permits history save-set display, but the equation scheme is not implemented.

- **Arguments**

- h : help
- h hostname : host name.
- e exptname : experiment name.
- D : start program as a daemon.
- u time : data update periodicity (def:5s).
- f file : file name (+.conf: if using existing file).
- q ODBpath : ODB tree path for extraction of the variables.
- c : ONLY creates the configuration file for later use.
- b lower\_value : sets general lower limit for all variables.
- t upper\_value : sets general upper limit for all variables.
- g : spawn the graphical stripchart if available.
- gg : force the use of gstripchart for graphic.
- gh : force the use of stripchart (tcl/tk) for graphic.

- **Usage** The configuration contains an entry for each variable found under the ODBpath requested. The format is described in the gstripchart documentation.

Once the configuration file has been created, it is possible to apply any valid operation (equation) to the parameters of the file following the gstripchart syntax.

In the case of the use of the *stripchart* from G.Hofman, only the "filename", "pattern", "maximum", "minimum" fields are used.

When using mchart with -D Argument, it is necessary to have the **MCHART\_DIR**(p. ??) defined in order to allow the daemon to find the location of the configuration and data files (see **Environment variables**(p. ??)).

```
chaos:~/chart> more trigger.conf
#Equipment:          >/equipment/kos_trigger/statistics
menu:                on
slider:              on
type:                gtk
```

```

minor_ticks:      12
major_ticks:      6
chart-interval:   1.000
chart-filter:     0.500
slider-interval:  0.200
slider-filter:    0.200
begin:            Events_sent
  filename:       /home/chaos/chart/trigger
  fields:         2
  pattern:        Events_sent
  equation:       \ $2
  color:          \ $blue
  maximum:        1083540.00
  minimum:        270885.00
  id_char:        1
end:              Events_sent
begin:            Events_per_sec.
  filename:       /home/chaos/chart/trigger
  fields:         2
  pattern:        Events_per_sec.
  equation:       $2
  color:          \ $red
  maximum:        1305.56
  minimum:        326.39
  id_char:        1
end:              Events_per_sec.
begin:            kBytes_per_sec.
  filename:       /home/chaos/chart/trigger
  fields:         2
  pattern:        kBytes_per_sec.
  equation:       $2
  color:          \ $brown
  maximum:        898.46
  minimum:        224.61
  id_char:        1
end:              kBytes_per_sec.

```

A second file (data file) will be updated a fixed interval by the *{mchart-}* utility.

```

chaos:~/chart> more trigger
Events_sent 6.620470e+05
Events_per_sec. 6.463608e+02
kBytes_per_sec. 4.424778e+02

```

- **Example**

- Creation with ODBpath being one array and one element of 2 sitting under variables/:

```

chaos:~/chart> mchart -f chvv -q /equipment/chv/variables/chvv -c
chaos:~/chart> ls -l chvv*
-rw-r--r-- 1 chaos users 474 Apr 18 14:37 chvv
-rw-r--r-- 1 chaos users 4656 Apr 18 14:37 chvv.conf

```

- Creation with ODBpath of all the sub-keys settings in variables:

```
mchart -e myexpt -h myhost -f chv -q /equipment/chv/variables -c
```

- Creation and running in debug:

```
chaos:~/chart> mchart -f chv -q /equipment/chv/variables -d
CHVV : size:68
#name:17 #Values:17
CHVI : size:68
```

- Running a pre-existing conf file (chv.conf) debug:

```
chaos:~/chart> mchart -f chv.conf -d
CHVV : size:68
#name:17 #Values:17
CHVI : size:68
#name:17 #Values:17
```

- Running a pre-existing configuration file and spawning **gstripchart**(p.??):

```
chaos:~/chart> mchart -f chv.conf -gg
spawning graph with gstripchart -g 500x200-200-800 -f /home/chaos/chart/chv.conf ...
```

- Running a pre-existing configuration file and spawning stripchart, this will work only if Tcl/Tk and bltwish packages are installed and the stripchart.tcl has been installed through the Midas Makefile.

```
chaos:~/chart> mchart -f chv.conf -gh
spawning graph with stripchart /home/chaos/chart/chv.conf ...
```

### 9.16.13 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
```

### 9.16.14 dio task

Direct I/O task provider (LINUX).

If no particular Linux driver is installed for the CAMAC access, the **dio-** program will allow you to 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 worst 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.

### 9.16.15 stripchart.tcl file

Graphical stripchart data display. Operates on **mchart task**(p.??) data or on Midas history save-set files. (see also **History system**(p.??)).

- **Arguments**

-mhist : start stripchart for Midas history data.

- **Usage** : stripchart <-options> <config-file> -mhist (look at history file -default) -dmhist debug mhist -debug debug stripchart config\_file: see mchart\_task

```
> stripchart.tcl -debug
> stripchart.tcl
```

- Example

```
> stripchart.tcl -h
```

gstripchart display with parameters and data pop-up.

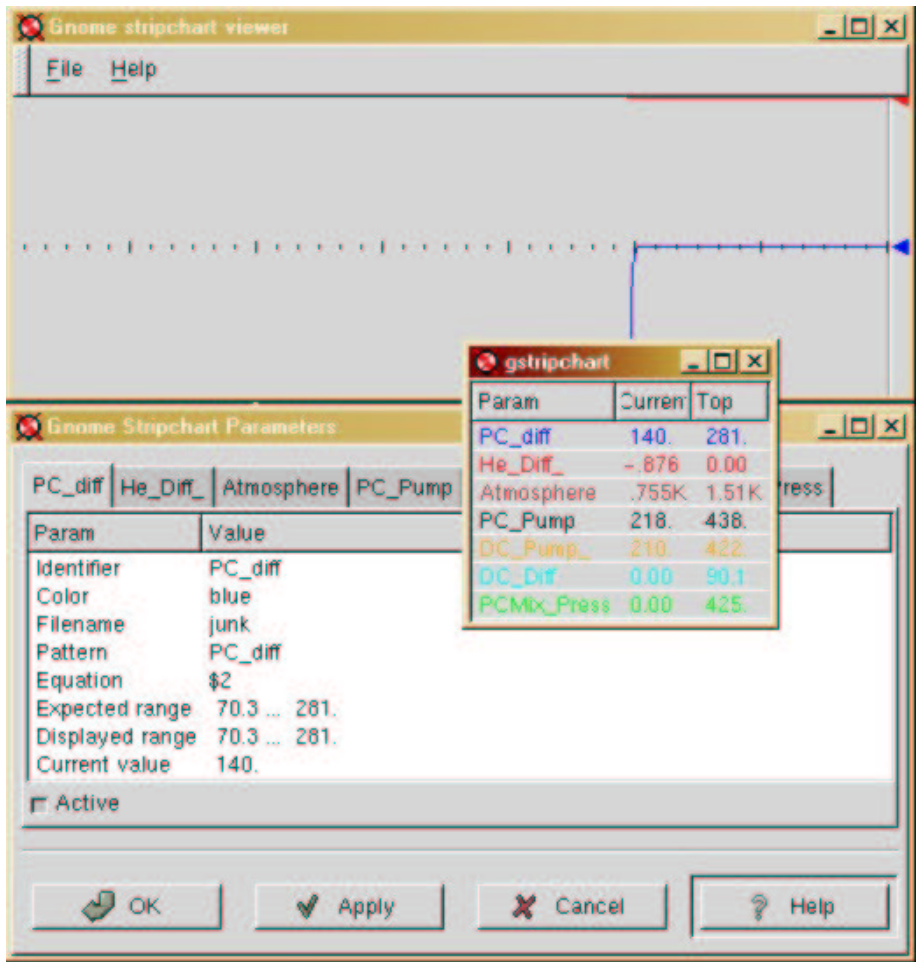


Figure 9.33: gstripchart display with parameters and data pop-up.

stripchart.tcl mhist mode: main window with pull-downs.

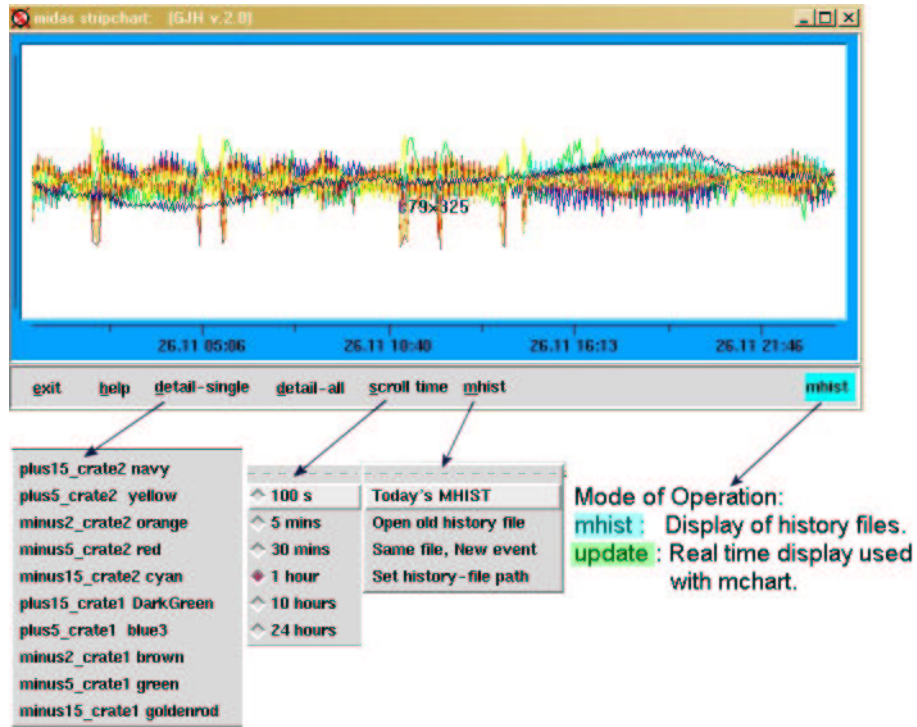


Figure 9.34: stripchart.tcl mhist mode: main window with pull-downs.

stripchart.tcl: Online data, running in conjunction with mchart.



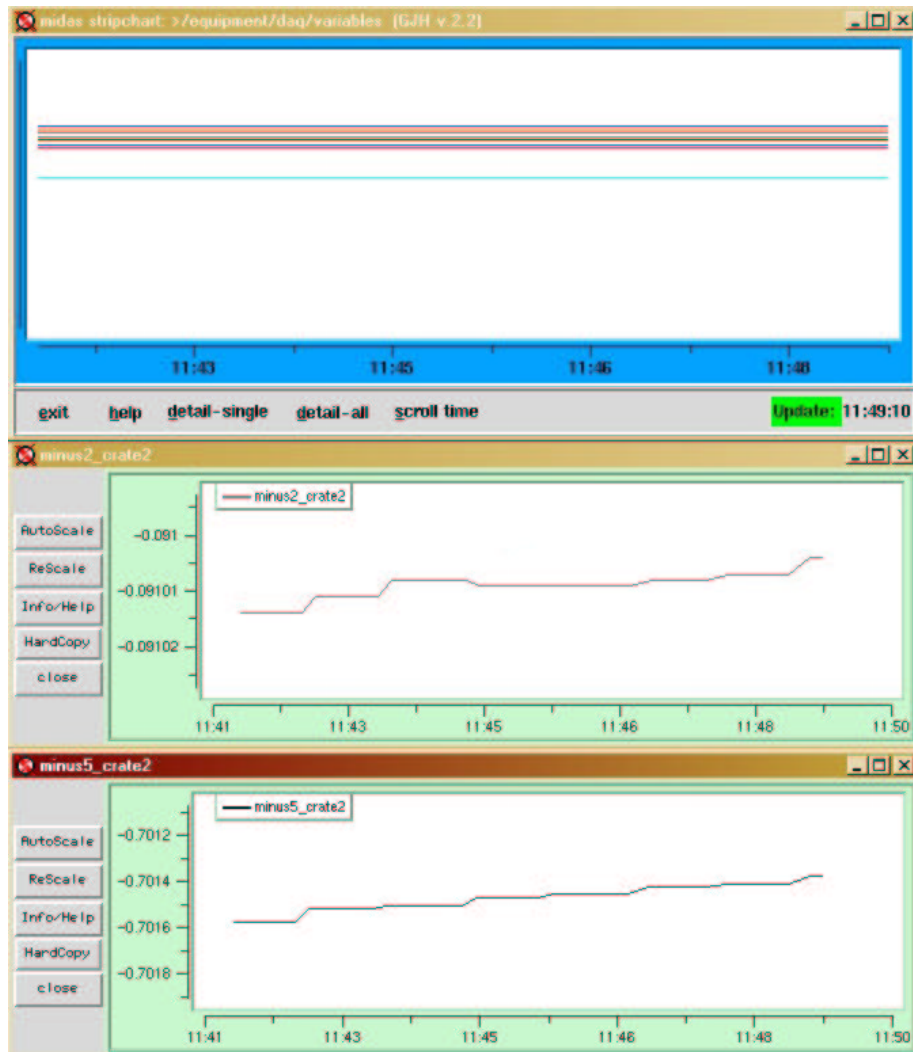


Figure 9.35: stripchart.tcl: Online data, running in conjunction with mchart.

### 9.16.16 rmidas task

Root/Midas remote GUI application for root histograms and 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
```

### 9.16.17 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
```