acontis technologies GmbH

SOFTWARE

EC-Daq

User Manual

Version 3.2

Edition: July 31, 2025

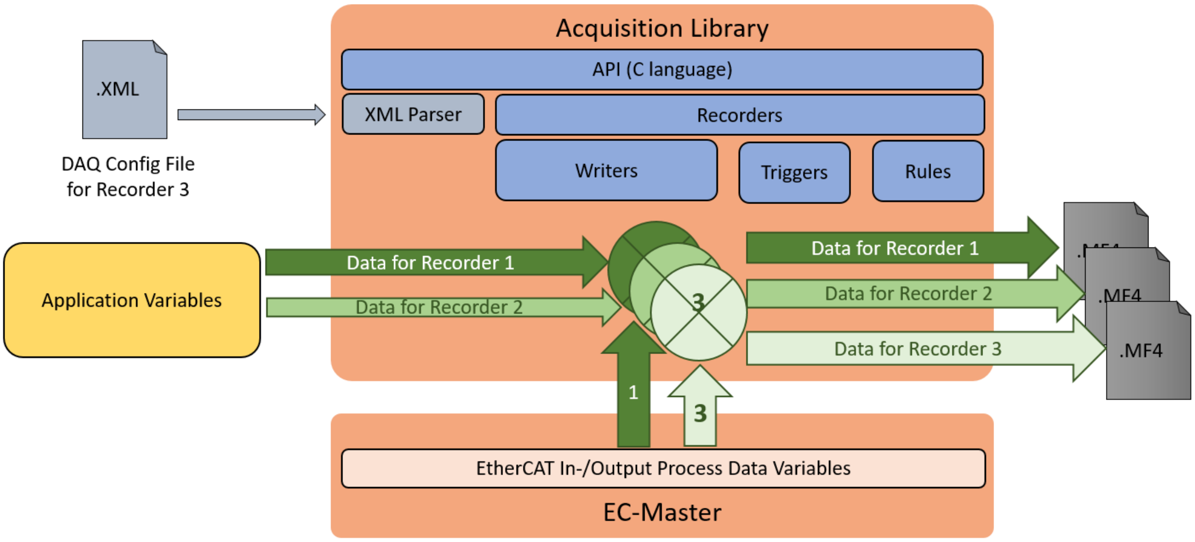
Contents

1 Introduction

1.1 Overview

The library *Data Acquisition* allows the application to record data real-time based on specific trigger conditions.

1.2 Architecture



The library exports a C-API for parameter setting and trigger conditions. It is possible to record data to multiple files concurrently, e. g. with different sample rates. The recorder can be also configured with a DAQ Config File (XML format).

1.3 Recorded Data

1.3.1 Process Variables

The library can record process variables which can be added by:

- Slave address
- Name of variable
- Data range with offset and length

1.3.2 Application Variables

The library can record application variables which can be registered from the application by submitting name, data type and address of the data buffer.

1.4 Trigger Recording

1.4.1 Manual

Recording can be started/stopped from application manually by calling appropriate API.

1.4.2 Automatic

Recording can be started/stopped automatically by adding triggers.

Currently two types are supported:

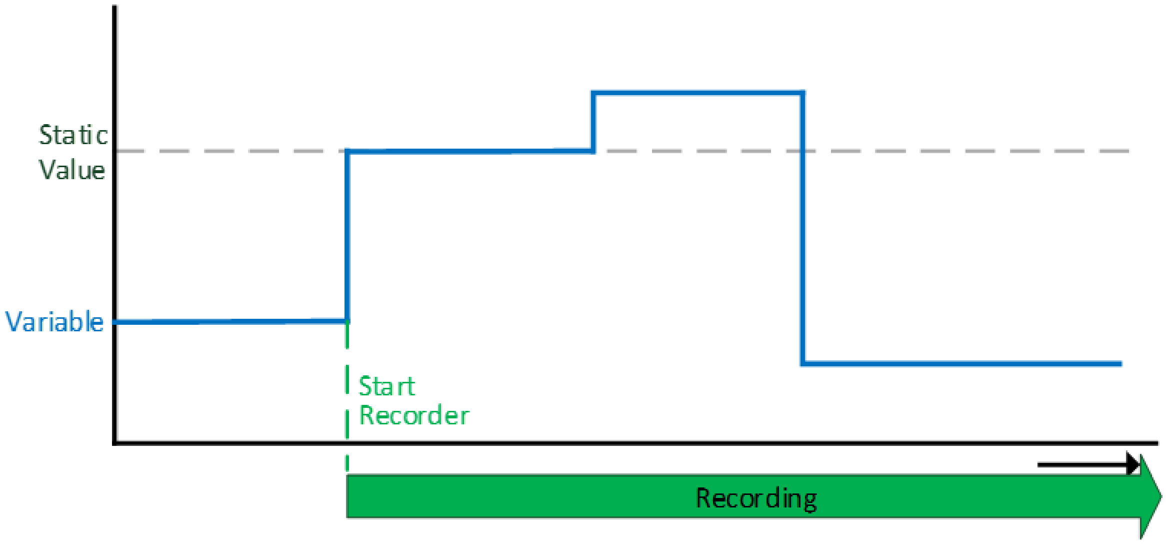
- Trigger by value
  - Variable will be compared against a static value
- Trigger by variable
  - Variable will be compared against another variable

Supported comparison operators:

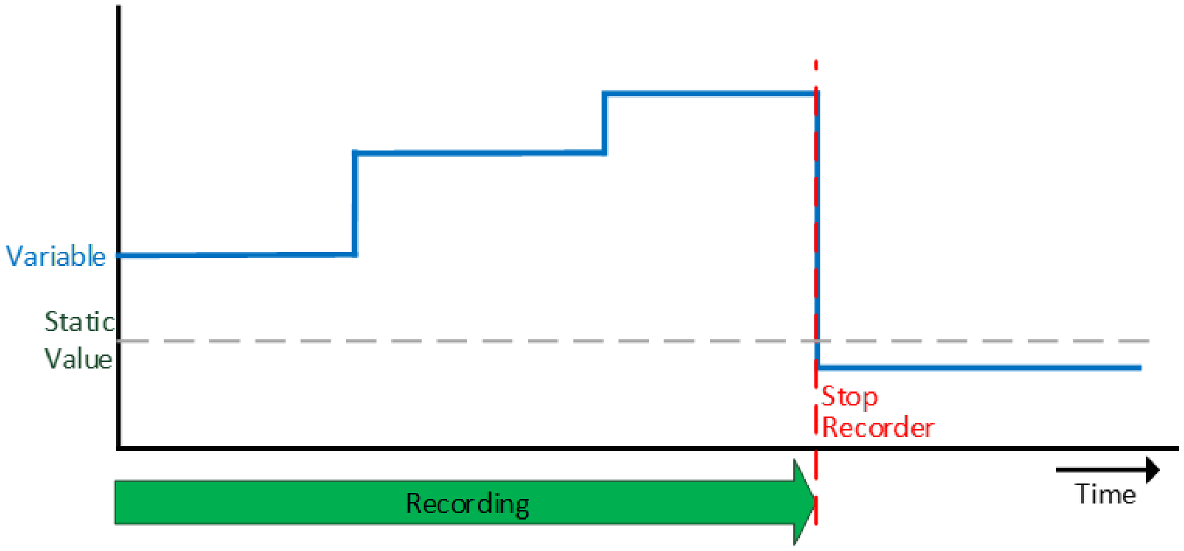
- Equal
- Greater
- Greater or equal
- Smaller
- Smaller or equal

Samples

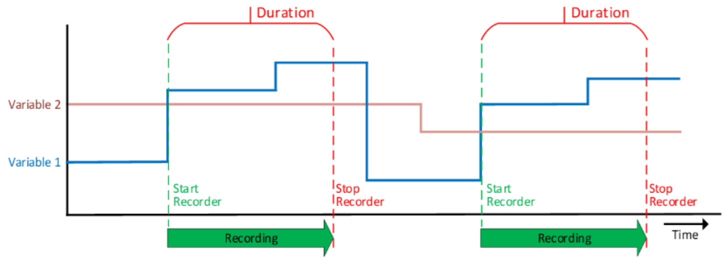
- Start recorder for infinite time, if variable >= static value



- Stop recorder, if variable < static value



- Start recorder for a specific duration, if “Variable 1” is greater than “Variable 2”



1.5 Supported File Formats

1.5.1 MDF (Measurement Data Format)

MDF (Measurement Data Format) is a binary file format for measurement data that was developed by Vector in collaboration with Robert Bosch GmbH in 1991.

After the MDF format quickly emerged as the de facto standard in the automotive industry, the revised Version 4.0 was ultimately published as an official ASAM standard in 2009.

The format was last updated as ASAM MDF 4.1 in 2012.

Find more at https://www.asam.net/standards/detail/mdf/

1.5.2 CSV (Comma Separated Values)

A comma-separated values (CSV) file is a delimited text file that uses a comma to separate values.

Find more at https://en.wikipedia.org/wiki/Comma-separated_values

1.6 Visualisation Software

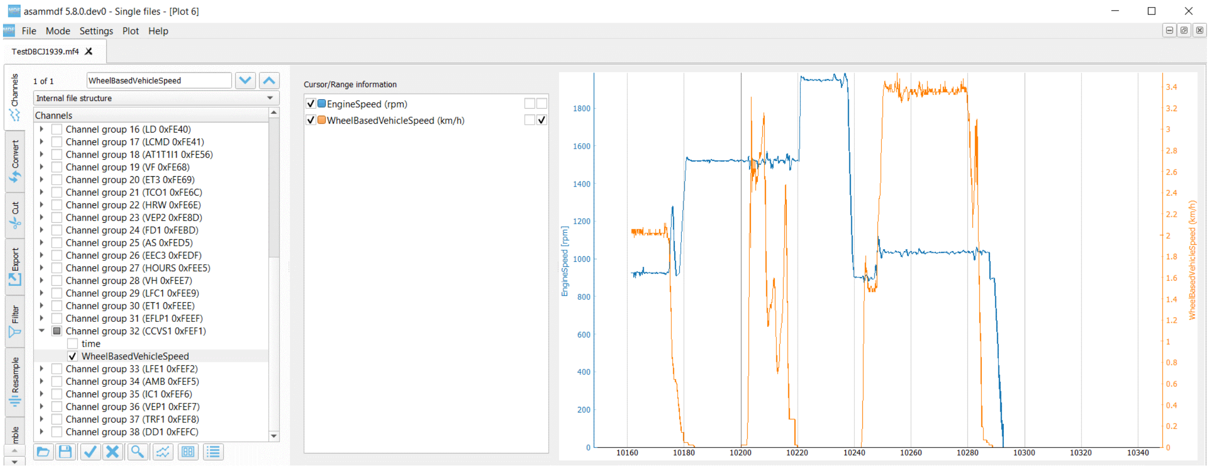
There are various tools available to analyse and evaluate the recorded data:

- Open source tool asammdf
- Vector vSignalyzer
- MathWorks MATLAB
- NI Labview with MDF4 DataPlugin
- Softing MDF.view
- Bosch WinDarab
- ETAS INCA
- Weisang FlexPro
- Imc FAMOS Reader

1.6.1 asammdf

asammdf is a fast parser and editor for ASAM (Associtation for Standardisation of Automation and Measuring Systems) MDF (Measurement Data Format) files.

asammdf supports MDF versions 2 (.dat), 3 (.mdf) and 4 (.mf4).



WheelBasedVehicleSpeed

Internal file structure

Channels

- □ Channel group 16 (LD 0xFE40)
- □ Channel group 17 (LCMD 0xFE41)
- □ Channel group 18 (AT1T1I1 0xFE56)
- □ Channel group 19 (VF 0xFE68)
- □ Channel group 20 (ET3 0xFE69)
- □ Channel group 21 (TCO1 0xFE6C)
- □ Channel group 22 (HRW 0xFE6E)
- □ Channel group 23 (VEP2 0xFE8D)
- □ Channel group 24 (FD1 0xFEBD)
- □ Channel group 25 (AS 0xFED5)
- □ Channel group 26 (EEC3 0xFEDF)
- □ Channel group 27 (HOURS 0xFEE5)
- □ Channel group 28 (VH 0xFEE7)
- □ Channel group 29 (LFC1 0xFEE9)
- □ Channel group 30 (ET1 0xFEEE)
- □ Channel group 31 (EFLP1 0xFEEF)
- ▨ Channel group 32 (CCVS1 0xFEF1)
  - □ time
  - ☑ WheelBasedVehicleSpeed
- □ Channel group 33 (LFE1 0xFEF2)
- □ Channel group 34 (AMB 0xFEF5)
- □ Channel group 35 (IC1 0xFEF6)
- □ Channel group 36 (VEP1 0xFEF7)
- □ Channel group 37 (TRF1 0xFEF8)
- □ Channel group 38 (DD1 0xFEFC)

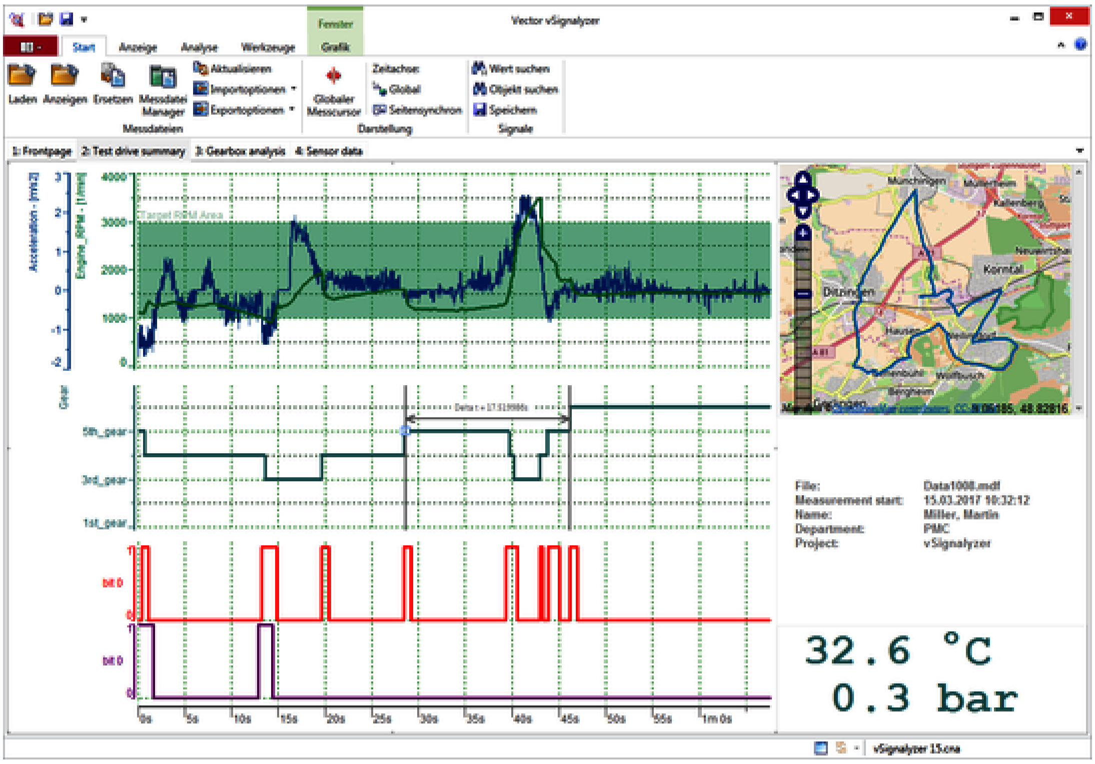
☑ EngineSpeed (rpm) □ □

☑ WheelBasedVehicleSpeed (km/h) □ ☑

For more information: https://pypi.org/project/asammdf/

1.6.2 vSignalizer

vSignalyzer is a convenient tool for efficiently evaluating measurement data of all types. It gives measurement engineers extensive options for visualizing the data as well as functions for manual and automated analysis and reporting.



File: Data1008.mdf
Measurement start: 15.03.2017 10:32:12
Name: Miller, Martin
Department: PMC
Project: vSignalyzer

32.6 °C
0.3 bar

For more information: http://www.vector.com/vSignalyzer

2 Getting Started

2.1 Running EcMasterDemoDaq

The EcMasterDemoDaq is available “out of the box” for different operating systems. It is an EC-Master example application that handles the following tasks:

- Showing basic EtherCAT communication
- Master stack initialization into OPERATIONAL state
- Process Data operations for e.g. Beckhoff EL2004, EL1004 and EL4132
- Periodic diagnosis task
- Periodic Job Task in polling mode
- Logging
- Record data

Start the EcMasterDemoDaq from the command line to put the EtherCAT network into operation. At least a Real-time Ethernet Driver must be specified.

```
$ EcMasterDemoDaq -ndis 192.168.157.2 1 -f eni.xml -t 0 -v 3 -daqrec Recorder.xml
```

2.1.1 Command line parameters

```
EcMasterDemoDaq <LinkLayer> [-f ENI-FileName] [-daqrec DAQ-CfgFile] [-
t time] [-b cycle time] [-a affinity] [-v lvl] [-perf] [-
log prefix [msg cnt]] [-lic key] [-oem key] [-maxbusslaves cnt] [-
flash address] [-printvars] [-sp [port]] [-rec [prefix [frame cnt]]]
```

The parameters are as follows:

-f `<ENI-FileName>`
Path to ENI file

-t `<time>`
Running duration in msec. When the time expires the demo application exits completely.

<time>
Time in msec, 0 = forever (default = 120000)

-b `<cycle time>`
Specifies the bus cycle time. Defaults to 1000 µs (1 ms).

<cycle time>
Bus cycle time in µsec

-a `<affinity>`
The CPU affinity specifies which CPU the demo application ought to use.

<affinity>
0 = first CPU, 1 = second, …

-v `<level>`
The verbosity level specifies how much console output messages will be generated by the demo application. A high verbosity level leads to more messages.

<level>
Verbosity level: 0=off (default), 1..n=more messages

-perf [<level>]
Enable max. and average time measurement in µs for all EtherCAT jobs (e.g. ProcessAllRxFrames).

<level>
Depending on level the performance histogram can be activated as well.

-log <prefix> [<msg cnt>]
Use given file name prefix for log files.

<prefix>

<msg cnt>
Messages count for log buffer allocation

-lic <key>
Set License key.

<key>
License key string

-oem <key>
Use OEM key

<key>
64 bit OEM key.

-maxbusslaves <cnt>
Set max number of slaves

-flash <address>
Flash outputs

<address>
0=all, >0 = slave station address

-sp [<port>]
If platform has support for IP Sockets, this command-line option enables the Remote API Server to be started. The Remote API Server is going to listen on TCP Port 6000 (or port parameter if given) and is available for connecting Remote API Clients.

<port>
RAS server port

-rec [<prefix> [<frame cnt>]]
Packet capture file recording

<prefix>
File name prefix

<frame cnt>
Frame count for log buffer allocation

-daqrec <FileName>

<FileName>
Configuration file

3 Application programming interface, reference

Function prototypes, definitions etc. of the API can be found in the header file `EcDaq.h` which is the main header file to include.

Fundamental types

typedef void ***EC_T_PVOID**
Pointer of type void

typedef int **EC_T_BOOL**
Boolean

typedef char **EC_T_CHAR**
Character, 8 bit

typedef unsigned short **EC_T_WCHAR**
Wide-character, 16 bit

typedef unsigned char **EC_T_BYTE**
Byte, unsigned integer 8 bit

typedef unsigned char ***EC_T_PBYTE**
Pointer of type EC_T_BYTE

typedef unsigned short **EC_T_WORD**
Word, unsigned integer 16 bit

typedef unsigned int **EC_T_DWORD**
Double word, unsigned integer 32 bit

typedef signed char **EC_T_SBYTE**
Signed-Byte, signed integer 8 bit

typedef signed short **EC_T_SWORD**
Signed-Word, signed integer 16 bit

typedef signed int **EC_T_SDWORD**
Signed-Double-Word, signed integer 32 bit

typedef int **EC_T_INT**
Integer

typedef unsigned int **EC_T_UINT**
Unsigned-Integer

typedef short **EC_T_SHORT**
Short

typedef unsigned short **EC_T_USHORT**
Unsigned-Short

typedef float **`EC_T_REAL`**
Real, floating point

typedef double **`EC_T_LREAL`**
long Real, floating point

`EC_T_VOID`
Void type

Macros

`EC_TRUE`
Boolean value: True

`EC_FALSE`
Boolean value: False

`EC_NULL`
Null pointer constant

3.1 Manage Recorders

Create or control recorders

3.1.1 ecDaqRecCreate

EC_T_DWORD **`ecDaqRecCreate`** (
EC_T_DAQ_REC *phDaq,
EC_T_DAQ_FP *pFp,
EC_T_DAQ_REC_PARMS *pParms
)

Create a DAQ recorder instance.

Parameters

- **`phDaq`** – [out] DAQ recorder handle
- **`pFp`** – [in] Function pointers
- **`pParms`** – [in] Parameters

Returns
EC_E_NOERROR on success, error code otherwise.

struct **`EC_T_DAQ_FP`**

Public Members

EC_T_DAQ_FP_GETVERSION **GetVersion**
[in] Returns version

EC_T_DAQ_FP_REGISTERCLIENT **RegisterClient**
[in] Register client

EC_T_DAQ_FP_UNREGISTERCLIENT **UnregisterClient**
[in] Unregister client

EC_T_DAQ_FP_GETCFGSLAVEINFO **GetCfgSlaveInfo**
[in] Returns config slave information

EC_T_DAQ_FP_GETSLAVEVARINFONUMOF **GetSlaveInpVarInfoNumOf**
[in] Returns number of input variables from slave

EC_T_DAQ_FP_GETSLAVEVARINFO **GetSlaveInpVarInfo**
[in] Returns input variables from slave

EC_T_DAQ_FP_GETSLAVEVARINFONUMOF **GetSlaveOutpVarInfoNumOf**
[in] Returns number of output variables from slave

EC_T_DAQ_FP_GETSLAVEVARINFO **GetSlaveOutpVarInfo**
[in] Returns output variables from slave

EC_T_DAQ_FP_GETAPPVARINFONUMOF **GetAppVarInfoNumOf**
[in] Returns number of application variables

EC_T_DAQ_FP_GETAPPVARINFO **GetAppVarInfo**
[in] Returns information about application variables

EC_T_DAQ_FP_SYSTEMTIMEGET **SystemTimeGet**
[in] Returns system time

EC_T_DAQ_FP_QUERYMSECCOUNT **QueryMsecCount**
[in] Query msec count

struct **EC_T_DAQ_REC_PARMS**

Public Members

EC_T_LOG_PARMS **LogParms**
[in] Logging parameters

EC_T_DWORD **dwMasterInstanceId**
[in] Master instance ID

EC_T_CHAR **szWriter**[64]
[in] Writer name (e.g. MDF or CSV)

EC_T_CHAR **szName**[128]
[in] Writer title

EC_T_CHAR **szFile**[256]
[in] Writer file name

EC_T_DWORD **dwSampleRate**
[in] Sample rate

EC_T_DWORD **dwBusCycleTimeUsec**
[in] Bus cycle time [us]

EC_T_BOOL **bRealTimeStamp**
[in] EC_TRUE: Real time stamp, EC_FALSE: Virtual time stamp

EC_T_BOOL **bCycleCounter**
[in] Cycle counter

EC_T_BOOL **bElapsedTimeMsec**
[in] Elapsed time [ms]

EC_T_BOOL **bElapsedTimeUsec**
[in] Elapsed time [ms]

EC_T_DWORD **dwLimitsMaxFileSize**
[in] Maximal file size in bytes

EC_T_DWORD **dwLimitsMaxDuration**
[in] Maximal duration in [ms]

EC_T_DWORD **dwLimitsMaxFiles**
[in] Maximal count of files

EC_T_DWORD **dwThreadMaxPendingDataSets**
[in] Maximal pending data sets

EC_T_DWORD **dwThreadCpuSet**
[in] CPU set of thread

EC_T_DWORD **dwThreadPrio**
[in] Priority of thread

EC_T_DWORD **dwThreadStackSize**
[in] Stack size of thread

EC_T_DWORD **dwOversamplingMaxRate**
[in] Maximal oversampling rate

EC_T_DWORD **dwMemoryCycleCount**
[in] Memory cycle count

3.1.2 ecDaqRecDelete

EC_T_DWORD **ecDaqRecDelete** (EC_T_DAQ_REC hDaq)

Delete the DAQ recorder instance.

Parameters

hDaq – [in] DAQ recorder handle

Returns

EC_E_NOERROR on success, error code otherwise.

3.1.3 ecDaqRecStart

EC_T_DWORD **ecDaqRecStart** (EC_T_DAQ_REC hDaq, *EC_T_DWORD* dwDuration = 0)

Start logging of recorder.

Parameters

- **hDaq** – [in] DAQ recorder handle
- **dwDuration** – [in] Duration [ms] after state should be changed (0 = infinite)

Returns

EC_E_NOERROR on success, error code otherwise.

3.1.4 ecDaqRecStartRt

EC_T_DWORD **ecDaqRecStartRt** (EC_T_DAQ_REC hDaq, *EC_T_DWORD* dwDuration = 0)

Start logging of recorder.

Must be called only from cyclic task.

Parameters

- **hDaq** – [in] DAQ recorder handle
- **dwDuration** – [in] Duration [ms] (0 = infinite)

Returns

EC_E_NOERROR on success, error code otherwise.

3.1.5 ecDaqRecStop

EC_T_DWORD **ecDaqRecStop** (EC_T_DAQ_REC hDaq, *EC_T_DWORD* dwDuration = 0)

Stop logging of recorder.

Parameters

- **hDaq** – [in] DAQ recorder handle
- **dwDuration** – [in] Duration [ms] after state should be changed (0 = infinite)

Returns

EC_E_NOERROR on success, error code otherwise.

3.1.6 ecDaqRecStopRt

EC_T_DWORD **ecDaqRecStopRt** (EC_T_DAQ_REC hDaq, *EC_T_DWORD* dwDuration = 0)

Stop logging of recorder.

Must be called only from cyclic task.

Parameters

- **hDaq** – [in] DAQ recorder handle
- **dwDuration** – [in] Duration [ms] after state should be changed (0 = infinite)

Returns

EC_E_NOERROR on success, error code otherwise.

3.1.7 ecDaqProcessRt

EC_T_DWORD **ecDaqProcessRt** (EC_T_DAQ_REC hDaq)

Process logging, called within cyclic task.

Parameters

hDaq – [in] DAQ recorder handle

Returns

EC_E_NOERROR on success, error code otherwise.

3.2 Configuration

Configure the recorder

3.2.1 ecDaqConfigLoad

EC_T_DWORD **ecDaqConfigLoad** (EC_T_DAQ_REC hDaq, const *EC_T_CHAR* *pszFile)

Loads configuration from file.

Parameters

- **hDaq** – [in] DAQ recorder handle
- **pszFile** – [in] Path to config file

Returns

EC_E_NOERROR on success, error code otherwise.

3.2.2 ecDaqConfigApply

EC_T_DWORD **ecDaqConfigApply** (EC_T_DAQ_REC hDaq)

Applies the configuration and prepares everything for start/stop logging.

Once the configuration is applied, it can no longer be changed. Logging can only be started or stopped. As a result, all “ecDaqConfig*” functions will return EC_E_INVALIDSTATE if the configuration has already been applied.

Parameters

hDaq – [in] DAQ recorder handle

Returns

EC_E_NOERROR on success, error code otherwise.

3.2.3 ecDaqConfigAddDataSlave

EC_T_DWORD **`ecDaqConfigAddDataSlave`** (EC_T_DAQ_REC hDaq, *EC_T_WORD* wAddress)

Add slave to logging.

Parameters

- **`hDaq`** – [in] DAQ recorder handle
- **`wAddress`** – [in] Station address of slave

Returns

EC_E_NOERROR on success, error code otherwise.

3.2.4 ecDaqConfigAddDataVariable

EC_T_DWORD **`ecDaqConfigAddDataVariable`** (
EC_T_DAQ_REC hDaq,
const *EC_T_CHAR* *pszName
)

Add variable to logging.

Parameters

- **`hDaq`** – [in] DAQ recorder handle
- **`pszName`** – [in] Name of slave

Returns

EC_E_NOERROR on success, error code otherwise.

3.2.5 ecDaqConfigAddDataRange

EC_T_DWORD **`ecDaqConfigAddDataRange`** (
EC_T_DAQ_REC hDaq,
EC_T_DWORD dwOffset,
EC_T_DWORD dwSize,
EC_T_BOOL bInput
)

Add process data range to logging.

Parameters

- **`hDaq`** – [in] DAQ recorder handle
- **`dwOffset`** – [in] Offset of data
- **`dwSize`** – [in] Size of data
- **`bInput`** – [in] EC_TRUE: input data, EC_FALSE: output data

Returns

EC_E_NOERROR on success, error code otherwise.

3.2.6 ecDaqConfigRegisterAppVariable

EC_T_DWORD **ecDaqConfigRegisterAppVariable** (
EC_T_DAQ_REC hDaq,
const *EC_T_CHAR* *pszName,
EC_T_WORD wDataType,
EC_T_INT nBitOffs,
EC_T_INT nBitSize,
EC_T_VOID *pvData
)

Register application variable to logging.

Parameters

- **hDaq** – [in] DAQ recorder handle
- **pszName** – [in] Name of variable
- **wDataType** – [in] Data type (see DEFTYPE_…)
- **nBitOffs** – [in] Bit offset of variable
- **nBitSize** – [in] Bit size of variable
- **pvData** – [in] Data pointer

Returns
EC_E_NOERROR on success, error code otherwise.

EtherCAT® data types

DEFTYPE_NULL
Null

DEFTYPE_BOOLEAN
Boolean, bit size: 1

DEFTYPE_INTEGER8
Integer, bit size: 8

DEFTYPE_INTEGER16
Integer, bit size: 16

DEFTYPE_INTEGER32
Integer, bit size: 32

DEFTYPE_UNSIGNED8
Unsigned, bit size: 8

DEFTYPE_UNSIGNED16
Unsigned, bit size: 16

DEFTYPE_UNSIGNED32
Unsigned, bit size: 32

DEFTYPE_REAL32
Real, bit size: 32

DEFTYPE_VISIBLESTRING
Visible string, bit size: 8*n

DEFTYPE_OCTETSTRING
Octet string, bit size: 8*(n+1)

DEFTYPE_UNICODESTRING
Unicode string, bit size: 16*(n+1)

DEFTYPE_TIMEOFDAY
Time of day

DEFTYPE_TIMEDIFFERENCE
Time difference

DEFTYPE_DOMAIN
Domain

DEFTYPE_INTEGER24
Integer, bit size: 24

DEFTYPE_REAL64
Real, bit size: 64

DEFTYPE_INTEGER40
Integer, bit size: 40

DEFTYPE_INTEGER48
Integer, bit size: 48

DEFTYPE_INTEGER56
Integer, bit size: 56

DEFTYPE_INTEGER64
Integer, bit size: 64

DEFTYPE_UNSIGNED24
Unsigned, bit size: 24

DEFTYPE_UNSIGNED40
Unsigned, bit size: 40

DEFTYPE_UNSIGNED48
Unsigned, bit size: 48

DEFTYPE_UNSIGNED56
Unsigned, bit size: 56

DEFTYPE_UNSIGNED64
Unsigned, bit size: 64

DEFTYPE_GUID
Guid, bit size: 128

DEFTYPE_BYTE
Byte, bit size: 8

DEFTYPE_WORD
Word, bit size: 16

DEFTYPE_DWORD
Dword, bit size: 32

DEFTYPE_PDOMAPPING
PDO Mapping

DEFTYPE_IDENTITY

DEFTYPE_COMMAND
Command

DEFTYPE_PDOCOMPAR
PDO COMPAR

DEFTYPE_ENUM
Enum

DEFTYPE_SMPAR
SMPAR

DEFTYPE_RECORD
Record

DEFTYPE_BACKUP_PARAMETER
Backup parameter

DEFTYPE_MODULAR_DEVICE_PROFILE
Modular device profile

DEFTYPE_BITARR8
Bit array, bit size: 8

DEFTYPE_BITARR16
Bit array, bit size: 16

DEFTYPE_BITARR32
Bit array, bit size: 32

DEFTYPE_BIT1
Bit, bit size: 1

DEFTYPE_BIT2
Bit, bit size: 2

DEFTYPE_BIT3
Bit, bit size: 3

DEFTYPE_BIT4
Bit, bit size: 4

DEFTYPE_BIT5
Bit, bit size: 5

DEFTYPE_BIT6
Bit, bit size: 6

DEFTYPE_BIT7
Bit, bit size: 7

DEFTYPE_BIT8
Bit, bit size: 8

DEFTYPE_BIT9
Bit, bit size: 9

DEFTYPE_BIT10
Bit, bit size: 10

DEFTYPE_BIT11
Bit, bit size: 11

DEFTYPE_BIT12
Bit, bit size: 12

DEFTYPE_BIT13
Bit, bit size: 13

DEFTYPE_BIT14
Bit, bit size: 14

DEFTYPE_BIT15
Bit, bit size: 15

DEFTYPE_BIT16
Bit, bit size: 16

DEFTYPE_ARRAY_OF_BYTE
Array of BYTE, bit size: 8 * (n + 1)

DEFTYPE_ARRAY_OF_UINT
Array of UINT, bit size: 16 * (n + 1)

DEFTYPE_ARRAY_OF_INT
Array of INT, bit size: 16 * (n + 1)

DEFTYPE_ARRAY_OF_SINT
Array of SINT, bit size: 8 * (n + 1)

DEFTYPE_ARRAY_OF_DINT
Array of DINT, bit size: 32 * (n + 1)

DEFTYPE_ARRAY_OF_UDINT
Array of UDINT, bit size: 32 * (n + 1)

DEFTYPE_ERROR_SETTING
Error setting, bit size: -

DEFTYPE_HISTORY
History, bit size: -

DEFTYPE_DIAGNOSIS_OBJECT
Diagnosis object, bit size: -

DEFTYPE_EXTERNAL_SYNC_STATUS
External SYNC status, bit size: -

DEFTYPE_EXTERNAL_SYNC_SETTINGS
External SYNC settings, bit size: -

DEFTYPE_FSOEFRAME
FSOE frame, bit size: -

DEFTYPE_FSOECOMMPAR
FSOE COMMPAR, bit size: -

3.3 Trigger

Control or configure a trigger of a recorder

3.3.1 ecDaqConfigAddTriggerByValue

EC_T_DWORD **ecDaqConfigAddTriggerByValue** (
EC_T_DAQ_REC hDaq,
EC_T_DAQ_TRIGGER *phTrigger,
const *EC_T_CHAR* *pszName,
const *EC_T_CHAR* *pszValue,
EC_T_DAQ_OPERATOR eOperator,
EC_T_BOOL bEnable,
EC_T_BOOL bStart,
EC_T_DWORD dwDuration = 0,
EC_T_DWORD dwCount = 0
)

Add trigger for starting/stopping logging by comparing a variable with a static value.

All trigger conditions will be evaluated in the order as they were added to the configuration. The first matched trigger will change the state of the recorder and duration and count of this trigger will be used.

Parameters

- **hDaq** – [in] DAQ recorder handle
- **phTrigger** – [out] Trigger handle
- **pszName** – [in] Name of variable
- **pszValue** – [in] Value to be compared (string representation)
- **eOperator** – [in] Operator
- **bEnable** – [in] EC_TRUE: enable trigger, can be disabled with *ecDaqTriggerDisable()*, EC_FALSE: disable trigger, can be enabled with *ecDaqTriggerEnable()*.

- **bStart** – [in] EC_TRUE: start recording after trigger condition matches, EC_FALSE: stop recording after trigger condition matches.
- **dwDuration** – [in] Maximum duration [ms] after recording state should be toggled (0 = infinite). bStart = EC_TRUE: If this start trigger condition matched and recorder currently is running, it will be stopped after dwDuration. bStart = EC_FALSE: If this trigger condition was true and recorder currently is stopped, it will be started after dwDuration elapsed.
- **dwCount** – [in] Trigger will be disabled after condition matches for dwCount times (0 = infinite). Such trigger will not be re-evaluated after disabling

Returns

EC_E_NOERROR on success, error code otherwise.

enum **EC_T_DAQ_OPERATOR**

Values:

enumerator **eDaqOperator_UNKNOWN**

Unknown

enumerator **eDaqOperator_Equal**

Equal

enumerator **eDaqOperator_Greater**

Greater

enumerator **eDaqOperator_GreaterOrEqual**

Greater or equal

enumerator **eDaqOperator_Smaller**

Smaller

enumerator **eDaqOperator_SmallerOrEqual**

Smaller or equal

enumerator **eDaqOperator_NotEqual**

Not equal

Examples

```
/* Recording is stopped. Recording will be started, if variable “myVariable” is␣
↪greater than 1: */

ecDaqConfigAddTriggerByValue(hDaq, EC_NULL, "myVariable", "1", eDaqOperator_
↪Greater, EC_TRUE /* bEnable */, EC_TRUE /* bStart */);
```

```
/* Recording is stopped. Recording will be started for 1000ms, if variable␣
↪“myVariable” is greater than 1 (after 1000ms recording will be automatically␣
↪stopped): */

ecDaqConfigAddTriggerByValue(hDaq, EC_NULL, "myVariable", "1", eDaqOperator_
↪Greater, EC_TRUE /* bEnable */, EC_TRUE /* bStart */, 1000 /* dwDuration */);
↪
```

```
/* Recording is stopped. Recording will be started for 1000ms, if variable␣
↪“myVariable” is greater than 1. This will happen for 10 times, afterwards this␣
↪trigger will be automatically disabled: */

```

(continues on next page)

(continued from previous page)

```
ecDaqConfigAddTriggerByValue(hDaq, EC_NULL, "myVariable", "1", eDaqOperator_
↪Greater, EC_TRUE /* bEnable */, EC_TRUE /* bStart */, 1000 /* dwDuration */,␣
↪10 /* dwCount */);
```

```
/* Recording is started. This will stop recording, if variable “myVariable” is␣
↪smaller than 1: */

ecDaqConfigAddTriggerByValue(hDaq, EC_NULL, "myVariable", "1", eDaqOperator_
↪Smaller, EC_TRUE /* bEnable */, EC_FALSE /* bStart */);
```

```
/* Recording is stopped.This will start recording, if “myVariable1” or␣
↪“myVariable2” is 1 and recording will be stopped, if “myVarible1” or␣
↪“myVarible2” is 0: */

ecDaqConfigAddTriggerByValue(hDaq, EC_NULL, "myVariable1", "1", eDaqOperator_Equal,
↪ EC_TRUE /* bEnable */, EC_TRUE /* bStart */);
ecDaqConfigAddTriggerByValue(hDaq, EC_NULL, "myVariable2", "1", eDaqOperator_Equal,
↪ EC_TRUE /* bEnable */, EC_TRUE /* bStart */);
ecDaqConfigAddTriggerByValue(hDaq, EC_NULL, "myVariable1", "0", eDaqOperator_Equal,
↪ EC_TRUE /* bEnable */, EC_FALSE /* bStart */);
ecDaqConfigAddTriggerByValue(hDaq, EC_NULL, "myVariable2", "0", eDaqOperator_Equal,
↪ EC_TRUE /* bEnable */, EC_FALSE /* bStart */);
```

3.3.2 ecDaqConfigAddTriggerByVariable

EC_T_DWORD **ecDaqConfigAddTriggerByVariable**(
EC_T_DAQ_REC hDaq,
EC_T_DAQ_TRIGGER *phTrigger,
const *EC_T_CHAR* *pszName1,
const *EC_T_CHAR* *pszName2,
EC_T_DAQ_OPERATOR eOperator,
EC_T_BOOL bEnable,
EC_T_BOOL bStart,
EC_T_DWORD dwDuration = 0,
EC_T_DWORD dwCount = 0
)

Add trigger for starting/stopping logging by comparing a variable with another variable (e.g. variable 1 is greater than variable 2).

Parameters

- **hDaq** – [in] DAQ recorder handle
- **phTrigger** – [out] Trigger handle
- **pszName1** – [in] Name of variable 1
- **pszName2** – [in] Name of variable 2
- **eOperator** – [in] Operator
- **bEnable** – [in] EC_TRUE: enable trigger, can be disabled with *ecDaqTriggerDisable()*, EC_FALSE: disable trigger, can be enabled with *ecDaqTriggerEnable()*.
- **bStart** – [in] EC_TRUE: start recording after trigger condition matches, EC_FALSE: stop recording after trigger condition matches.
- **dwDuration** – [in] Maximum duration [ms] after recording state should be toggled (0 = infinite). bStart = EC_TRUE: If this start trigger condition matched and recorder currently is running, it will be stopped after dwDuration. bStart = EC_FALSE: If this trigger

condition was true and recorder currently is stopped, it will be started after dwDuration elapsed.

- **dwCount** – [in] Trigger will be disabled after condition matches for dwCount times (0 = infinite). Such trigger will not be re-evaluated after disabling

Returns

EC_E_NOERROR on success, error code otherwise.

Example

```
/* Recording is stopped. This will start recording for 1000ms, if variable␣
↪"myVariable1" is greater than variable "myVariable2": */

ecDaqConfigAddTriggerByVariable(hDaq,  EC_NULL, "myVariable1", "myVariable2",␣
↪eDaqOperator_Greater, EC_TRUE, EC_TRUE, 1000);

/* Compare also samples of "ecDaqConfigAddTriggerByVariable()". */
```

3.3.3 ecDaqTriggerEnable

EC_T_DWORD **ecDaqTriggerEnable** (EC_T_DAQ_REC hDaq, EC_T_DAQ_TRIGGER hTrigger)

Enable trigger.

Parameters

- **hDaq** – [in] DAQ recorder handle
- **hTrigger** – [in] Trigger handle

Returns

EC_E_NOERROR on success, error code otherwise.

3.3.4 ecDaqTriggerDisable

EC_T_DWORD **ecDaqTriggerDisable** (
EC_T_DAQ_REC hDaq,
EC_T_DAQ_TRIGGER hTrigger
)

Disable trigger.

Parameters

- **hDaq** – [in] DAQ recorder handle
- **hTrigger** – [in] Trigger handle

Returns

EC_E_NOERROR on success, error code otherwise.

3.3.5 ecDaqTriggerGetCount

EC_T_DWORD **ecDaqTriggerGetCount** (

    EC_T_DAQ_REC hDaq,
    EC_T_DAQ_TRIGGER hTrigger,
    EC_T_DWORD *pdwCount
)

Returns trigger count.

Parameters

- **hDaq** – [in] DAQ recorder handle
- **hTrigger** – [in] Trigger handle
- **pdwCount** – [out] Trigger count

Returns

EC_E_NOERROR on success, error code otherwise.

4 Examples

4.1 Initialization

```
EC_T_DAQ_REC S_hDaq = EC_NULL;

EC_T_DAQ_FP tEcDaqFp;
OsMemset(&tEcDaqFp, 0, sizeof(EC_T_DAQ_FP));
tEcDaqFp.RegisterClient = emRegisterClient;
tEcDaqFp.UnregisterClient = emUnregisterClient;
tEcDaqFp.GetCfgSlaveInfo = emGetCfgSlaveInfo;
tEcDaqFp.GetSlaveInpVarInfoNumOf = emGetSlaveInpVarInfoNumOf;
tEcDaqFp.GetSlaveInpVarInfo = emGetSlaveInpVarInfoEx;
tEcDaqFp.GetSlaveOutpVarInfoNumOf = emGetSlaveOutpVarInfoNumOf;
tEcDaqFp.GetSlaveOutpVarInfo = emGetSlaveOutpVarInfoEx;
tEcDaqFp.SystemTimeGet = OsSystemTimeGet;
tEcDaqFp.QueryMsecCount = OsQueryMsecCount;

EC_T_DAQ_REC_PARMS tEcDaqParms;
OsMemset(&tEcDaqParms, 0, sizeof(EC_T_DAQ_REC_PARMS));
tEcDaqParms.dwMasterInstanceId = dwInstanceId;
tEcDaqParms.dwSampleRate = 0;
tEcDaqParms.dwBusCycleTimeUsec = 1000;
tEcDaqParms.bRealTimeStamp = EC_FALSE;
tEcDaqParms.bCycleCounter = EC_TRUE;
tEcDaqParms.bElapsedTimeMsec = EC_TRUE;
strcpy(tEcDaqParms.szWriter, "MDF");
strcpy(tEcDaqParms.szName, "WriterMdf.mf4");
strcpy(tEcDaqParms.szFile, "c:\\temp\\WriterMdf.mf4");

dwRes = ecDaqRecCreate(&S_hDaq, &tEcDaqFp, &tEcDaqParms);
```

4.2 Configuration

```
dwRes = ecDaqConfigAddDataVariable(hDaq, "Slave_1005.Inputs.Channel");
dwRes = ecDaqConfigApply(hDaq);
```

4.3 Start

```
dwRes = ecDaqRecStart(hDaq);
```

4.4 Stop

```
dwRes = ecDaqRecStop(hDaq);
```

4.5 Processing

```
dwRes = ecDaqProcessRt(hDaq);
```

4.6 Deinitialization

```
dwRes = ecDaqRecDelete(hDaq);
```

5 FAQ

Can the library be used without EC-Master?

Yes, in that case the all callbacks must be implemented without using EC-Master.

Will the file be closed after calling `ecDaqRecStop()`?

No, the file will be closed after deleting the recorder instance by calling `ecDaqRecDelete()`.

What does happen if internal buffer is too small?

The library will report an error with the logger and the message will be deleted.

The buffer can be increased by changing *`EC_T_DAQ_REC_PARMS::dwThreadMaxPendingDataSets`* (default: 1000).

If this doesn't help, maybe the recorder threads needs to run on higher priority *`EC_T_DAQ_REC_PARMS::dwThreadPrio`*.