UeiDaq Framework Reference Manual
Revision 1.8.0

January 2020 Edition

Contents

Chapter 1

UeiDaq Framework Main Page

1.1 Introduction

The UeiDaq framework provides a complete object oriented programming interface including a class hierarchy designed to configure and control data acquisition or generation over Analog or Digital I/O.

The support for different devices is done through plug-ins loaded at runtime. The library comes with plug-ins for PCI and PXI devices of the PowerDAQ family, ethernet devices of the PowerDNA family and Simulation devices.

1.2 Features

- Support for C, C++, ActiveX, Java and all languages supported by the .NET framework
- URL like resource selection
- Simple data transfer timed by software
- Buffered data transfer timed by hardware
- Advanced timing configuration: external clock, multi devices synchronization
- Advanced trigger configuration

1.3 Compiling a C++ program

To compile a program that use the UeiDaq library you have to add the following line at the top of your program:

```
#include <UeiDaq.h>
```

1.4 Compiling a .NET program

Add a reference to the UeiDaqDNet assembly in your project references. you can then access the classes of the library with their full name or with their short name if you add the following at the

top of your program:

```
using UeiDaq;
```

1.5 Selecting a resource

The plugin used to communicate with a device is dynamically selected depending on the resource string.

- simu://Dev0/ai0 simulation device 0, AI channel 0.
- pwrdaq://Dev1/ai0:7 PowerDAQ board 1, AI channels 0 to 7.
- pdna://192.168.100.2/Dev2/ci2 PowerDNA I/O module at address 192.168.100.2, device 2, Counter 2.

1.6 Programming with UeiDaq

The main class is the session class **CUeiSession**. It allows configuration of the channel list and control of the session.

```
CUeiSession *pSession = new CUeiSession();
```

Once created, the session object needs to be initialized with the resource string so that it knows which device and subsystem is should use.

```
pSession->CreateAIChannel("pwrdaq://Dev1/ai0", min, max, mode);
```

The timing paraneters are stored in a CUeiTiming object that is a member of the session object. It can be configured "manually" with its accessor methods or more simply with a method of the session object.

```
pSession->ConfigureTimingForBufferedIO(nbSamplesPerChannel, source, rate, edge, duration);
```

Data acquired or generated by a session are represented by a stream object. The data can be accessed directly through the stream object but it is easier to use a reader or writer object to access the stream.

```
CUeiAnalogScaledReader *pReader = new CUeiAnalogScaledReader(pSession->GetDataStream());
```

The reader automatically starts the session when read is called.

```
pReader->ReadMultipleScans(nbScans, buffer);

pSession->Stop();
```

Chapter 2

UeiDaq Framework Class Documentation

2.1 _tUeiANSITime Struct Reference

Structure represents time in C ANSI format with the addition of microseconds.

`#include <UeiStructs.h>`

Public Attributes

- **tm tm**
  C ANSI time structure.
- **uInt32 microseconds**
  Number of microseconds within the last second.

2.1.1 Detailed Description

Structure represents time in C ANSI format with the addition of microseconds

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.2 _tUeiAOWaveformParameters Struct Reference

Structure represents waveform shape parameters.

`#include <UeiStructs.h>`

Public Attributes

- **tUeiAOWaveformType type**
  waveform type/shape
- **tUeiAOWaveformMode mode**
  waveform mode affects timing
- **tUeiAOWaveformXform xform**
  waveform xform affects shape
- double **frequency**
  waveform frequency
- double **span**
  waveform span
- double **offset**
  waveform offset
- double **phase**
  waveform phase
- double **applyTime**
  apply parameters gradually over time interval
- float **dutyCycle**
  the duty cycle (for pulse waveform only)
- float **riseTime**
  the time it takes to transition from low to high state, % [0..1] (for pulse and sawtooth only)
- float **fallTime**
  the time it takes to transition from high to low state, % [0..1] (for pulse and sawtooth only)

2.2.1 Detailed Description

Structure used to update waveform shape parameters on the fly.

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.3 _tUeiAOWaveformSweepParameters Struct Reference

Structure represents waveform sweep parameters.

`#include <UeiStructs.h>`

Public Attributes

- **tUeiAOWaveformSweepControl control**
  sweep control
- **tUeiTimingDuration mode**
  sweep mode, continuous or single shot
- **uInt32 stepsUp**
  number of steps used to sweep up during N periods
- **uInt32 stepsDown**
  number of steps used to sweep down during N periods
- **uInt32 numberOfPeriods**
  Specify duration of the sweep in number of periods.
- double **lowerFrequency**
  lower frequency
- double **upperFrequency**
  upper frequency
- double **lowerAmplitude**
  lower amplitude
- double **upperAmplitude**
  upper amplitude
- double **lowerOffset**
  lower offset
- double **upperOffset**
  upper offset
- double **lowerPhase**
  lower phase
- double **upperPhase**
  upper phase
- double **sweepTime**
  Specify duration of the sweep in ms.

2.3.1 Detailed Description

Structure used to update waveform sweep parameters on the fly.

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.4 _tUeiARINCFilterEntry Struct Reference

Structure represents an ARINC filter entry.

`#include <UeiStructs.h>`

Public Attributes

- **uInt32 Label**

  Label to accept.

- **Int32 NewData**

  Accept word only if it carries different data from a previous received one.

- **Int32 TriggerSchedulerEntry**

  Trigger the scheduler entry with the same index as this filter entry in the scheduler table of the output port that has the same index as this input port.

2.4.1 Detailed Description

Structure represents an ARINC filter entry. ARINC frames whose label doesn't match any of the filter entry are discarded.

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.5 _tUeiARINCSchedulerEntry Struct Reference

Structure represents an ARINC scheduler entry.

`#include <UeiStructs.h>`

Public Attributes

- **Int32 Master**

  Specifies whether this is a master or slave entry All slave entries following a master entry (until the next master entry) are scheduled at the same time.

- **Int32 Periodic**

  Specifies whether this master entry should be scheduled periodically Ignored for slave entries.

- **uInt32 Delay**

  Sets time interval in us for the output of master entries.

- **tUeiARINCWord Word**

  Word to be sent when this entry is processed.

- **uInt32 MinorFrameMask**

  Minor frame mapping (one bit per frame).

2.5.1 Detailed Description

Structure represents an ARINC scheduler entry. The scheduler is used to emit ARINC frames periodically or when a specified frame is received. Only Master entries are scheduled followed by any adjacent slave entries.

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.6 _tUeiARINCWord Struct Reference

Structure represents an ARINC word.

`#include <UeiStructs.h>`

Public Attributes

- **uInt32 Label**

  The label of the word. It is used to determine the data type of the Data field, therefore, the method of data translation to use.

- **uInt32 Ssm**

  Sign/Status Matrix or SSM. This field contains hardware equipment condition, operational mode, or validity of data content.

- **uInt32 Sdi**

  Source/Destination Identifier or SDI. This is used for multiple receivers to identify the receiver for which the data is destined.

- **uInt32 Parity**

  The parity bit. The parity bit is automatically set on transmitted words. The parity bit on received words is actually a parity status. It is set to 0, when parity is odd and the receiver counts an odd number of 1s (all Ok). It is set to 1, when parity is odd and the receiver counts an even number of 1s (parity error) It is set to 1, when parity is even and the receiver counts an even number of 1s (all Ok). It is set to 0, when parity is even and the receiver counts an odd number of 1s (parity error).

- **uInt32 Data**

  The payload of the word. Its format depends on the label. Most common formats are BCD (binary-coded-decimal) encoding, BNR (binary) encoding or discrete format where each bit represents a Pass/Fail, True/False or Activated/Non-Activated condition.

2.6.1 Detailed Description

Structure represents an ARINC word

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.7 _tUeiBCDTime Struct Reference

Structure represents time in BCD format.

`#include <UeiStructs.h>`

Public Attributes

- **uInt32 microseconds**
  Number of microseconds within the last second.
- **uInt32 s100**
  100's of a second (4 bits)
- **uInt32 s10**
  10's of a second (4 bits)
- **uInt32 seconds**
  seconds (7 bits)
- **uInt32 minutes**
  minutes (7 bits)
- **uInt32 hours**
  hours (7 bits)
- **uInt32 days**
  days
- **uInt32 years**
  years

2.7.1 Detailed Description

Structure represents time in BCD (Binary Coded Decimal) format

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.8 _tUeiCanFilterEntry Struct Reference

Structure represents a CAN filter entry.

`#include <UeiStructs.h>`

Public Attributes

- **uInt32 First**

  First ID that delimits the range of IDs to accept.

- **uInt32 Last**

  Last ID that delimits the range of IDs to accept.

2.8.1 Detailed Description

Each filter entry is a range of arbitration IDs IDs greater or equal to first and less or equal to last will pass through All others will be rejected

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.9 _tUeiCanFrame Struct Reference

Structure represents a CAN frame.

`#include <UeiStructs.h>`

Public Attributes

- **uInt32 Id**
  CAN Frame arbitration id.
- **tUeiCANFrameType Type**
  Specifies whether this is a data, remote or error frame.
- **uInt32 DataSize**
  The number of significant bytes in the payload.
- **uInt8 Data** [8]
  The frame's payload. It can contain up to 8 bytes.

2.9.1 Detailed Description

Structure represents a CAN frame

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.10 _tUeiI2CMasterCommand Struct Reference

Parameters for a master command.

`#include <UeiStructs.h>`

Public Attributes

- **tUeiI2CCommand type**

  Type of I2C command for master to transmit.

- **Int32 slaveAddress**

  Address of slave to send command to.

- **uInt8 data** [255]

  Data for write command.

- **Int32 numWriteElements**

  Number of elements in data for write command.

- **Int32 numReadElements**

  Number of elements to request for read command.

- **tUeiI2CSlaveAddressWidth slaveAddressWidth**

  Send command as 7 or 10-bit address command. If slaveAddress is greater than 7-bit, 10-bit will be used by default.

2.10.1 Detailed Description

Structure specifies parameters for a master command

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.11 _tUeiI2CMasterMessage Struct Reference

Data stored by and read from master.

`#include <UeiStructs.h>`

Public Attributes

- **uInt8 stopBit**
  Bit to determine if this message is the last from an I2C transaction.
- **uInt8 data**
  8-bit data

2.11.1 Detailed Description

Structure specifies format for data read from master

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.12 _tUeiI2CSlaveMessage Struct Reference

Data stored by and read from slave.

`#include <UeiStructs.h>`

Public Attributes

- **tUeiI2CBusCode busCode**
  Context of the data or status code.
- **uInt8 data**
  8-bit data

2.12.1 Detailed Description

Structure specifies format for data read from slave

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.13 _tUeiMIL1553A708ControlFrame Struct Reference

Structure represents an ARINC-708 control frame (W/O).

`#include <UeiStructs.h>`

Inherited by **UeiDaq::CUeiMIL1553A708ControlFrame**.

Public Attributes

- **tUeiMIL1553A708Ops Operation**
  Operation.
- double **MinorClock**
  major clock rate: 0 == software clock
- double **MajorClock**
  minor clock rate: 0 == software clock
- **uInt32 MajorParam**
  major parameter
- **uInt32 MinorParam**
  minor parameter

2.13.1 Detailed Description

Structure represents an ARINC-708 control frame

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.14 _tUeiMIL1553A708DataFrame Struct Reference

Structure represents an ARINC-708 data frame (W/R).

`#include <UeiStructs.h>`

Inherited by **UeiDaq::CUeiMIL1553A708DataFrame**.

Public Attributes

- **uInt32 Timestamp**
  Timestamp in 10us resolution.
- **uInt32 Available**
  Available bumber of 16-bit words in the FIFO after the operation.
- **uInt32 DataSize**
  Data Size (returned on read, uses declared frame size on write).
- **uInt16 data** [100]
  ARINC-708 data.

2.14.1 Detailed Description

Structure represents a an ARINC-708 data frame

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.15 _tUeiMIL1553BCCBDataFrame Struct Reference

Structure represents a MIL-1553 BCCB (W/O).

`#include <UeiStructs.h>`

Inherited by **UeiDaq::CUeiMIL1553BCCBDataFrame**.

Public Attributes

- **uInt16 MNFrame**
  Minor frame # for this BCCB.
- **uInt16 Index**
  Index in the minor frame.
- **uInt16 Block**
  I/O block frame belongs to.
- **uInt16 flags0**
  FLAGS0 word in the BC control block.
- **uInt16 flags1**
  FLAGS1 word in the BC control block.
- **uInt16 flags2**
  Delay in us before command execution (bit15 is enable).
- **uInt16 flags3**
  Reserved for future flags.
- **uInt16 cmd1**
  First 1553 command word.
- **uInt16 cmd2**
  Second 1553 command word.
- **uInt16 sts1_or**
  "OR" mask for the first status word
- **uInt16 sts1_and**
  "AND" mask for the first status word
- **uInt16 sts1_val**
  Compare "VALUE" for the first status word.
- **uInt16 sts2_or**
  "OR" mask for the second status word
- **uInt16 sts2_and**

"AND" mask for the second status word

- **uInt16 sts2_val**
  Compare "VALUE" for the second status word.
- **uInt16 rx_data** [32]
  RX data (from BC to RT) or minimum compare values for TX.
- **uInt16 tmin** [32]
  minimum compare values for TX
- **uInt16 tmax** [32]
  maximum compare values for TX

2.15.1 Detailed Description

Structure represents a 1553 frame to write BCCB data

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.16 _tUeiMIL1553BCCBStatusFrame Struct Reference

Structure represents a MIL-1553 read BCCB status (R/O).

`#include <UeiStructs.h>`

Inherited by **UeiDaq::CUeiMIL1553BCCBStatusFrame**.

Public Attributes

- **uInt16 MNFrame**
  Minor frame # for this BCCB.
- **uInt16 Index**
  Index in the minor frame.
- **uInt16 Block**
  I/O block frame belongs to.
- **uInt16 sts1**
  First status reply from the RT.
- **uInt16 sts2**
  Second status reply from the RT.
- **uInt16 tsmsb**
  Timestamp of last access to the RT, 16 MSBs.
- **uInt16 tslsb**
  Timestamp of last access to the RT, 16 LSBs.
- **uInt16 errsts0**
  error status 0
- **uInt16 errsts1**
  error status 1
- **uInt16 tx_data** [32]
  transmit data (from RT to BC)

2.16.1 Detailed Description

Structure represents a 1553 frame to read BCCB status

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.17 _tUeiMIL1553BCControlFrame Struct Reference

Structure represents a MIL-1553 BCCB control frame (W/O).

`#include <UeiStructs.h>`

Inherited by **UeiDaq::CUeiMIL1553BCControlFrame**.

Public Attributes

- **tUeiMIL1553BCOps Operation**
  Operation.
- double **MinorClock**
  major clock rate: 0 == software clock
- double **MajorClock**
  minor clock rate: 0 == software clock
- **uInt32 MajorParam**
  major parameter
- **uInt32 MinorParam**
  minor parameter

2.17.1 Detailed Description

Structure represents a MIL-1553 BCCB control frame

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.18 _tUeiMIL1553BCSchedFrame Struct Reference

Structure represents a MIL-1553 BCCB scheduler frame (W/R).

`#include <UeiStructs.h>`

Inherited by **UeiDaq::CUeiMIL1553BCSchedFrame**.

Public Attributes

- **tUeiMIL1553BCFrameType FrameType**
  Frame type (major or minor).
- **uInt16 Block**
  I/O block terminal frame belongs to.
- **uInt32 FrameIndex**
  Frame index.
- **uInt32 DataOffset**
  First entry to write to (offset).
- **uInt32 DataSize**
  The number of words in the payload.
- **uInt32 data** [36]
  The frame payload. It can contain up to 36 uInt32s (content of entries).

2.18.1 Detailed Description

Structure represents a MIL-1553 BCCB scheduler frame

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.19 _tUeiMIL1553BCStatusFrame Struct Reference

Structure represents a MIL-1553 BC status (R/O).

`#include <UeiStructs.h>`

Inherited by **UeiDaq::CUeiMIL1553BCStatusFrame**.

Public Attributes

- **uInt32 BCStatus**
  BC status word.
- **uInt32 PortStatus**
  Port status word.
- **uInt32 MJPos**
  Current MJ position.
- **uInt32 MNPos**
  Current minor position.
- **uInt32 Block**
  Current block.
- **uInt32 IsrStatus**
  BC ISR status word.
- **uInt32 ErrStatus**
  BC ERR status word.

2.19.1 Detailed Description

Structure represents a MIL-1553 BC status

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.20 _tUeiMIL1553BMCmdFrame Struct Reference

Structure represents a MIL-1553 BM frame with decoded command sequence (R/O).

`#include <UeiStructs.h>`

Inherited by **UeiDaq::CUeiMIL1553BMCmdFrame**.

Public Attributes

- **uInt8 Bus**

  Which bus the message was received on; 1 = A, 0 = B.

- **uInt16 Rt**

  Remote terminal frame belongs to.

- **uInt16 Sa**

  Sub-address frame belongs to.

- **uInt16 WordCount**

  Number of data words in the <Command> or status word from the bus or Mode Code.

- **tUeiMIL1553CommandType Command**

  Command type.

- **uInt16 Rt2**

  Second Remote terminal, in the case of an RT-RT command.

- **uInt16 Sa2**

  Second Sub-address, in the case of an RT-RT command.

- **uInt16 Status**

  First status.

- **uInt16 Status2**

  Second status, in the case of an RT-RT command.

- **uInt32 Timestamp**

  Timestamp of message start, in tens of microseconds.

- **uInt32 Delay**

  Delay in us before the command or timestamp.

- **Int8 DataWordStartIdx**

  An index into <BmData> where the first data word of the message is located, or -1 if the message has no data. If a Mode Code command, the number of data words is 1; otherwise, <WordCount> indicates the data word count (where 0 means 32).

- **uInt32 BmDataSize**

  The number of words that make up the entire frame (i.e. the size of the <BmData> field).

- **uInt32 BmData** [42]

  The frame, bus monitor and TxFIFO frame payload. It can contain up to 42 uInt32s: 32 points of data plus 2 commands 2 statuses and 3 timestamps and 3 flags.

2.20.1 Detailed Description

Structure represents a MIL-1553 BM frame with decoded command sequence (R/O)

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.21 _tUeiMIL1553BMFrame Struct Reference

Structure represents a bus monitor MIL-1553 frame (R/O).

`#include <UeiStructs.h>`

Inherited by **UeiDaq::CUeiMIL1553BMFrame**.

Public Attributes

- **tUeiMIL1553CommandType Command**

  Command type.

- **uInt16 Rt**

  Remote terminal frame belongs to.

- **uInt16 Sa**

  Sub-address frame belongs to.

- **uInt16 WordCount**

  Number of data words in the <Command> or status word from the bus or Mode Code.

- **uInt16 CmdStatusRaw**

  First command status.

- **uInt32 Flags**

  Flags if requested.

- **uInt32 Delay**

  Delay in us before the command or timestamp.

- **uInt32 DataSize**

  The number of bytes in the payload (also size of data from BusWriter frame).

- **uInt32 BmData** [36]

  The frame, bus monitor and TxFIFO frame payload. It can contain up to 36 uInt32s - command/status, 32 points of data, timestamp and flags.

2.21.1 Detailed Description

Structure represents a 1553 frame

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.22 _tUeiMIL1553BusWriterFrame Struct Reference

Structure represents a MIL-1553 bus writer data (W/O).

`#include <UeiStructs.h>`

Inherited by **UeiDaq::CUeiMIL1553BusWriterFrame**.

Public Attributes

- **uInt16 Rt**
  Remote terminal frame belongs to.
- **uInt16 Sa**
  Sub-address frame belongs to.
- **uInt16 WordCount**
  Number of data words in the <Command> or status word from the bus or Mode Code.
- **tUeiMIL1553CommandType Command**
  Command type.
- **uInt16 Rt2**
  Remote terminal frame belongs to.
- **uInt16 Sa2**
  Sub-address frame belongs to.
- **uInt32 Flags**
  Flags if requested.
- **uInt32 Delay**
  Delay in us before the command or timestamp.
- **uInt32 DataSize**
  The number of bytes in the payload (also size of data from BusWriter frame).
- **uInt32 TxData** [36]
  The frame, bus monitor and TxFIFO frame payload. It can contain up to 36 uInt32s.

2.22.1 Detailed Description

Structure represents a 1553 frame to write the bus

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.23 _tUeiMIL1553Command Struct Reference

Structure represents a MIL-1553 command.

`#include <UeiStructs.h>`

Public Attributes

- **tUeiMIL1553CommandType Command**
  Command type.
- **uInt16 Rt2**
  Second terminal for UeiMIL1553CmdRTRT type.
- **uInt16 Sa2**
  Second subaddress for UeiMIL1553CmdRTRT type.
- **uInt16 WordCount**
  Number of data words in the <Command> or status word from the bus or Mode Code.
- **uInt32 Flags**
  Flags if requested.
- **uInt32 Delay**
  Delay in us before the command or timestamp.

2.23.1 Detailed Description

Structure represents a 1553 command

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.24 _tUeiMIL1553FilterEntry Struct Reference

Structure represents an 1553 filter entry.

`#include <UeiStructs.h>`

Inherited by **UeiDaq::CUeiMIL1553FilterEntry**.

Public Attributes

- **tUeiMIL1553FilterType FilterType**
  Type of the filtering to apply.
- int **RemoteTerminal**
  Remote terminal to allow.
- int **SubAddress**
  Subaddress to allow.
- int **MinRxLength**
  Minimal RT message data length for Rx command (0=32 words, -1=Any).
- int **MaxRxLength**
  Maximum RT message data length for Rx command (0=32 words, -1=Any).
- int **MinTxLength**
  Minimal RT message data length for Tx command (0=32 words, -1=Any).
- int **MaxTxLength**
  Maximum RT message data length for Tx command (0=32 words, -1=Any).
- int **EnableRxRequests**
  Enable Rx requests.
- int **EnableTxRequests**
  Enable Tx requests.
- int **EnableModeCommands**
  Enable mode commands or Raw validation table value.

2.24.1 Detailed Description

Structure represents an 1553 filter entry. 1553 messages outside of enabled are ignored

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.25 _tUeiMIL1553Frame Struct Reference

Structure represents a MIL-1553 frame.

`#include <UeiStructs.h>`

Public Attributes

- **tUeiMIL1553FrameType Type**
  Specifies the type of 1553 frame.
- **uInt16 Rt**
  Remote terminal frame belongs to.
- **uInt16 Sa**
  Sub-address frame belongs to.
- **uInt16 Block**
  I/O block terminal frame belongs to.
- **tUeiMIL1553Command Command**
  Command is required for BC functionality (for UeiMIL1553FrameTypeBusWriter).
- **uInt32 DataSize**
  The number of bytes in the payload (also size of data from BusWriter frame).
- **uInt32 RxTxData** [36]
  The frame, bus monitor and TxFIFO frame payload. It can contain up to 36 uInt32s.

2.25.1 Detailed Description

Structure represents a 1553 frame (W/R)

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.26 _tUeiMIL1553RTControlFrame Struct Reference

Structure represents a MIL-1553 RT control data (W/O).

`#include <UeiStructs.h>`

Inherited by **UeiDaq::CUeiMIL1553RTControlFrame**.

Public Attributes

- **uInt16 Rt**
  Remote terminal frame belongs to.
- **uInt16 Sa**
  Subaddress frame belongs to (optional).
- **tUeiMIL1553RTControlType Flags**
  Flags, what control data is.
- **uInt32 BlockSelectRtBc**
  Which Tx (RT->BC) block to select.
- **uInt32 BlockSelectBcRt**
  Which Rx (BC->RT) block to select.
- **uInt32 Control0**
  Control 0 field.
- **uInt32 Control1**
  Control 1 field.
- **uInt32 GapBetweenDataWords**
  Defines rt response timing.

2.26.1 Detailed Description

Structure represents a 1553 frame to write control RT data

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.27 _tUeiMIL1553RTFrame Struct Reference

Structure represents a MIL-1553 RT frame Data (W/R).

`#include <UeiStructs.h>`

Inherited by **UeiDaq::CUeiMIL1553RTFrame**.

Public Attributes

- **uInt16 Rt**
  Remote terminal frame belongs to.
- **uInt16 Sa**
  Sub-address frame belongs to.
- **uInt16 Block**
  I/O block terminal frame belongs to.
- **uInt32 DataSize**
  The number of bytes in the payload (also size of data from BusWriter frame).
- **uInt32 Flags**
  Status flags associated with this data.
- **uInt32 RxTxData** [36]
  The frame, bus monitor and TxFIFO frame payload. It can contain up to 32 uInt32s.

2.27.1 Detailed Description

Structure represents a 1553 frame to read/write RT data

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.28 _tUeiMIL1553RTParametersFrame Struct Reference

Structure represents a MIL-1553 RT parameter (W/O).

`#include <UeiStructs.h>`

Inherited by **UeiDaq::CUeiMIL1553RTParametersFrame**.

Public Attributes

- **uInt16 Rt**
  Remote terminal frame belongs to.
- **uInt16 Sa**
  Sub-address frame belongs to.
- **uInt32 Flags**
  Flags, what control data it is; possibly we need emumerator.
- **uInt32 Parameter**
  Parameter type.
- **uInt32 DataSize**
  Size of data to transmit.
- **uInt32 Data** [32]
  Parameter data.

2.28.1 Detailed Description

Structure represents a 1553 frame to write RT parameters

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.29 _tUeiMIL1553RTStatusFrame Struct Reference

Structure represents a MIL-1553 RT status data (R/O).

`#include <UeiStructs.h>`

Inherited by **UeiDaq::CUeiMIL1553RTStatusFrame**.

Public Attributes

- **uInt16 Rt**
  Remote terminal frame belongs to.
- **uInt32 BusStatus**
  Remote tesminal bus status.
- **uInt32 DataReady**
  Bitmask for SAs which received data from BC (Rx command).
- **uInt32 DataSent**
  Bitmask for SAs which sent data to BC or another RT (Tx command).
- **uInt32 Status0**
  Status 0 field.
- **uInt32 Status1**
  Status 1 field.
- **uInt32 ChStatus**
  Channel status.

2.29.1 Detailed Description

Structure represents a 1553 frame to read status RT data

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.30 _tUeiMIL1553RTStatusFrameExt Struct Reference

Structure extends MIL-1553 RT status data (R/O).

`#include <UeiStructs.h>`

Public Attributes

- **uInt32 Status2**
  Status 2 field.
- **uInt32 Status3**
  Status 3 field.

2.30.1 Detailed Description

This structure extends tUeiMIL1553RTStatusFrame with extra status words

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.31 _tUeiMIL1553SchedulerEntry Struct Reference

Structure represents an 1553 scheduler entry.

`#include <UeiStructs.h>`

Public Attributes

- **Int32 Master**

  Specifies whether this is a master or slave entry.

- **Int32 Periodic**

  Specifies whether this entry should be scheduled periodically.

- **uInt32 Delay**

  Scheduling delay count in us.

- **tUeiMIL1553CommandType Command**

  Command to send.

- **uInt32 RemoteTerminal**

  Remote terminal to to send to.

- **uInt32 SubAddress**

  Subaddress to send to.

- **uInt32 EntrySize**

  Word to be sent when this entry is processed.

2.31.1 Detailed Description

Structure represents an 1553 scheduler entry. The scheduler is used to emit 1553 frames periodically or when a specified frame is received. Only Master entries are scheduled followed by any adjacent slave entries.

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.32 _tUeiPTPTime Struct Reference

Structure represents time in secs and nsecs.

`#include <UeiStructs.h>`

Public Attributes

- **uInt32 sec**

  Number of seconds since 01/01/1970.

- **uInt32 nsec**

  Number of nanoseconds elapsed before next second.

2.32.1 Detailed Description

Structure represents time in seconds since UNIX epoch time and nanoseconds remainder

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.33 _tUeiSBSTime Struct Reference

Structure represents time in SBS format.

`#include <UeiStructs.h>`

Public Attributes

- **uInt32 seconds**
  Number of seconds since beginning of the day.
- **uInt32 microseconds**
  Number of microseconds within the last second.
- **uInt32 dayofyear**
  day of the year
- **uInt32 year**
  year

2.33.1 Detailed Description

Structure represents time in SBS (Straight Binary Seconds) format

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.34 _tUeiSignal Struct Reference

Structure represents a synchronization signal.

`#include <UeiStructs.h>`

Public Attributes

- **tUeiSignalSource Source**
  Source of the signal.
- int **Device**
  *Device where this signal is located (used only if source is "*Trigger" or "*Clock").*
- int **Index**
  Index of the signal (used only if source is "Backplane").
- int **Mode**
  Mode enables device dependant timing modes.

2.34.1 Detailed Description

Structure represents a synchronization signal

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.35 _tUeiVRData Struct Reference

Structure represents variable reluctance data.

`#include <UeiStructs.h>`

Public Attributes

- double **velocity**
  velocity of the encoder wheel
- **uInt32 position**
  position as tooth index relative to the Z index tooth
- **uInt32 teethCount**
  number of teeth detected since last read
- **uInt32 timestamp**
  timestamp of measurement
- **uInt32 flags**
  status flags

2.35.1 Detailed Description

Structure used to store data measured on variable reluctance input devices.

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.36 _UeiAOShortOpenCircuitParameters Struct Reference

Outout channel short/open simulation.

`#include <UeiStructs.h>`

Public Attributes

- unsigned int **openBitmask**
  open circuit bitmask
- unsigned int **shortBitmask**
  short circuit bitmask
- unsigned int **shortDuration**
  amount of time in us that the output(s) will stay shorted (0 to stay shorted)

2.36.1 Detailed Description

This structure specifes which channel should be shorted or opened

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.37 _UeiCSDBMessage Struct Reference

CSDB message.

`#include <UeiStructs.h>`

Public Attributes

- unsigned char **address**
  address byte for the message
- unsigned char **status**
  status byte for the message
- unsigned char **data** [10]
  data bytes for the message. Number of bytes is block size - 2

2.37.1 Detailed Description

This structure represents a CSDB message

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.38 _UeiSync1PPSPTPStatus Struct Reference

Structure used to read PTP status.

`#include <UeiStructs.h>`

Public Attributes

- **tUeiPTPState State**
  PTP state.
- **uInt64 GrandMasterClockID**
  PTP clock ID of the grand master of the system.
- **uInt64 MasterClockID**
  PTP clock ID of the current master.
- **uInt32 StepsFromGrandMaster**
  The value of the PTP data set stepsRemoved.
- **uInt32 GrandMasterClockClass**
  The PTP grand master clock class taken from the master's announce messages.
- **Int32 MeanPathDelay**
  Current calculated mean path delay.
- **Int32 LastMeasuredOffset**
  Last calculated time offset from master.
- **Int32 MaxMeasuredOffset**
  Maximum calculated time offset from master.
- **Int32 MinMeasuredOffset**
  Minimum calculated time offset from master.
- **Int32 AvgMeasuredOffset**
  Average calculated time offset from master.

2.38.1 Detailed Description

This structure represents PTP status

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.39 _UeiSync1PPSStatus Struct Reference

Structure used to read ADPLL, event module and sync lines status.

`#include <UeiStructs.h>`

Public Attributes

- **uInt32 ADPLLStatus**
  ADPLL status register.
- **uInt32 ADPLLMinInputClockPeriod**
  Minimum period length for the input clock.
- **uInt32 ADPLLAvgInputClockPeriod**
  Averaged detected length of the VALIDATED input period.
- **uInt32 ADPLLMaxInputClockPeriod**
  Maximum period length for the input clock.
- **uInt32 ADPLLLastInputClockPeriod**
  Measured length of the last input period.
- **uInt32 ADPLLAccumulatedError**
  Accumulated pulse position error in system clocks.
- **uInt32 PLLStatus**
  whether PLLs are locked (8347/5200 "classic" have only one PLL)
- **uInt32 PPSStatus**
  status of nPPS signal
- **uInt32 EventModuleStatus**
  event module status
- **uInt32 SyncLinesStatus**
  snapshot of the SYNC lines
- **uInt32 SyncConnectorStatus**
  snapshot of the SyncIn/SyncOut lines

2.39.1 Detailed Description

This structure represents 1PPS synchronization hardware status

The documentation for this struct was generated from the following file:

- UeiStructs.h

2.40 UeiDaq::CUeiAccelChannel Class Reference

Manages settings for each accelerometer input channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiAIChannel**.

Public Member Functions

- UeiDaqAPI **CUeiAccelChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiAccelChannel** ()
  Destructor.
- UeiDaqAPI **tUeiCoupling GetCouplingType** ()
  Get the coupling type.
- UeiDaqAPI void **SetCouplingType** (**tUeiCoupling** coupling)
  Set the coupling type.
- UeiDaqAPI bool **IsLowPassFilterEnabled** ()
  Get the low-pass filter state.
- UeiDaqAPI void **EnableLowPassfilter** (bool enabled)
  Enable or Disable the low-pass filter.
- UeiDaqAPI double **GetSensorSensitivity** ()
  Get the sensor sensitivity.
- UeiDaqAPI void **SetSensorSensitivity** (double sensitivity)
  Set the sensor sensitivity.
- UeiDaqAPI double **GetExcitationCurrent** ()
  Get the excitation current.
- UeiDaqAPI void **SetExcitationCurrent** (double excCurrent)
  Set the excitation current.
- UeiDaqAPI double **GetLowExcitationComparator** ()
  Get the excitation low comparator voltage.
- UeiDaqAPI void **SetLowExcitationComparator** (double lowComparator)
  Set the excitation low comparator voltage.
- UeiDaqAPI double **GetHighExcitationComparator** ()
  Get the excitation high comparator voltage.
- UeiDaqAPI void **SetHighExcitationComparator** (double highComparator)

Set the excitation high comparator voltage.

- UeiDaqAPI bool **GetLowAlarmStatus** ()
  Get the low alarm status.
- UeiDaqAPI bool **GetHighAlarmStatus** ()
  Get the high alarm status.

2.40.1 Detailed Description

Manages settings for each accelerometer input channel This type of channel works with IEPE sensors and microphones

2.40.2 Member Function Documentation

2.40.2.1 UeiDaqAPI tUeiCoupling UeiDaq::CUeiAccelChannel::GetCouplingType ()

Get the channel coupling type, AC or DC

Returns:

the Coupling type.

See also:

SetCouplingType (p. 46)

2.40.2.2 UeiDaqAPI void UeiDaq::CUeiAccelChannel::SetCouplingType (tUeiCoupling *coupling*)

Configure the channel in AC or DC coupling.

Parameters:

coupling The coupling type

See also:

GetCouplingType (p. 46)

2.40.2.3 UeiDaqAPI bool UeiDaq::CUeiAccelChannel::IsLowPassFilterEnabled ()

Get the low-pass filter state

Returns:

the low pass filter state.

See also:

EnableLowPassfilter (p. 47)

2.40.2.4 UeiDaqAPI void UeiDaq::CUeiAccelChannel::EnableLowPassfilter (bool *enabled*)

Enable or disable the low-pass filter.

Parameters:

enabled The low-pass filter state

See also:

IsLowPassFilterEnabled (p. 46)

2.40.2.5 UeiDaqAPI double UeiDaq::CUeiAccelChannel::GetSensorSensitivity ()

Get the sensor sensitivity, it usually is in mVolts/[EU] Independently of the Engineering Unit (EU), the voltage read will be converted to mV and divided by the sensitivity

Returns:

the current sensitivity.

See also:

SetSensorSensitivity (p. 47)

2.40.2.6 UeiDaqAPI void UeiDaq::CUeiAccelChannel::SetSensorSensitivity (double *sensitivity*)

Set the sensor sensitivity, it usually is in mVolts/[EU] Independently of the Engineering Unit (EU), the voltage read will be converted to mV and divided by the sensitivity

Parameters:

sensitivity The new sensitivity

See also:

GetSensorSensitivity (p. 47)

2.40.2.7 UeiDaqAPI double UeiDaq::CUeiAccelChannel::GetExcitationCurrent ()

Get the excitation current in mAmps

Returns:

the excitation current.

See also:

SetExcitationCurrent (p. 48)

2.40.2.8 UeiDaqAPI void UeiDaq::CUeiAccelChannel::SetExcitationCurrent (double *excCurrent*)

Set the excitation current in mAmps

Parameters:

excCurrent The new excitation current

See also:

GetExcitationCurrent (p. 47)

2.40.2.9 UeiDaqAPI double UeiDaq::CUeiAccelChannel::GetLowExcitationComparator ()

The excitation current can be measured back and trigger an alarm when it is out of range (due to an open connection or a short). It is measured as a voltage which depends on the sensor resistance and the programmed excitation current. The alarm state is made visible with the LED color on the terminal block Green means that the measured excitation is within range, red means that the connection with the sensor is open and blinking red means that the connection with the sensor is shorted.

Returns:

the low excitation comparator voltage

See also:

SetLowExcitationComparator (p. 48), **GetLowAlarmStatus** (p. 49)

2.40.2.10 UeiDaqAPI void UeiDaq::CUeiAccelChannel::SetLowExcitationComparator (double *lowComparator*)

The excitation current can be measured back and trigger an alarm when it is out of range (due to an open connection or a short). It is measured as a voltage which depends on the sensor resistance and the programmed excitation current. The alarm state is made visible with the LED color on the terminal block Green means that the measured excitation is within range, red means that the connection with the sensor is open and blinking red means that the connection with the sensor is shorted.

Parameters:

lowComparator the new low excitation comparator voltage

See also:

GetLowExcitationComparator (p. 48), **GetLowAlarmStatus** (p. 49)

2.40.2.11 UeiDaqAPI double UeiDaq::CUeiAccelChannel::GetHighExcitationComparator ()

The excitation current can be measured back and trigger an alarm when it is out of range (due to an open connection or a short). It is measured as a voltage which depends on the sensor resistance

and the programmed excitation current. The alarm state is made visible with the LED color on the terminal block Green means that the measured excitation is within range, red means that the connection with the sensor is open and blinking red means that the connection with the sensor is shorted.

Returns:

the high excitation comparator voltage

See also:

SetHighExcitationComparator (p. 49), **GetHighAlarmStatus** (p. 49)

2.40.2.12 UeiDaqAPI void UeiDaq::CUeiAccelChannel::SetHighExcitationComparator (double *highComparator*)

The excitation current can be measured back and trigger an alarm when it is out of range (due to an open connection or a short). It is measured as a voltage which depends on the sensor resistance and the programmed excitation current. The alarm state is made visible with the LED color on the terminal block Green means that the measured excitation is within range, red means that the connection with the sensor is open and blinking red means that the connection with the sensor is shorted.

Parameters:

highComparator the new high excitation comparator voltage

See also:

GetHighExcitationComparator (p. 48), **GetHighAlarmStatus** (p. 49)

2.40.2.13 UeiDaqAPI bool UeiDaq::CUeiAccelChannel::GetLowAlarmStatus ()

Return the low alarm status which indicates whether the connection to the sensor is shorted. It also is visible on the terminal block as a red blinking LED.

Returns:

the low alarm state

See also:

SetLowExcitationComparator (p. 48)

2.40.2.14 UeiDaqAPI bool UeiDaq::CUeiAccelChannel::GetHighAlarmStatus ()

Return the high alarm status which indicates whether the connection to the sensor is open. It also is visible on the terminal block as a steady red LED.

Returns:

the high alarm state

See also:

SetHighExcitationComparator (p. 49)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.41 UeiDaq::CUeiAIChannel Class Reference

Manages settings for each voltage Analog input channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Inherited by **UeiDaq::CUeiAccelChannel**, **UeiDaq::CUeiAICurrentChannel**, **UeiDaq::CUeiAIVExChannel**, **UeiDaq::CUeiDMMChannel**, **UeiDaq::CUeiLVDTChannel**, **UeiDaq::CUeiResistanceChannel**, **UeiDaq::CUeiSynchroResolverChannel**, and **UeiDaq::CUeiTCChannel**.

Public Member Functions

- UeiDaqAPI **CUeiAIChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiAIChannel** ()
  Destructor.
- UeiDaqAPI **f64 GetMinimum** ()
  Get the minimum value.
- UeiDaqAPI void **SetMinimum** (**f64** minimum)
  Set the minimum value.
- UeiDaqAPI **f64 GetMaximum** ()
  Get the maximum value.
- UeiDaqAPI void **SetMaximum** (**f64** maximum)
  Set the maximum value.
- UeiDaqAPI **tUeiAIChannelInputMode GetInputMode** ()
  Get the input mode.
- UeiDaqAPI void **SetInputMode** (**tUeiAIChannelInputMode** mode)
  Set the input mode.
- UeiDaqAPI **f64 GetGain** ()
  Get the gain.
- UeiDaqAPI void **SetGain** (**f64** gain)
  Set the gain.
- UeiDaqAPI CUeiCustomScale * **GetCustomScale** ()
  Get custom scale.
- UeiDaqAPI void **SetCustomScale** (CUeiCustomScale *pScale)
  Set custom scale.
- UeiDaqAPI bool **IsOpenCircuitTestEnabled** ()

Get open circuit detection status.

- UeiDaqAPI void **EnableOpenCircuitTest** (bool enable)
  Enable or disable open cicuit detection.
- UeiDaqAPI bool **IsCircuitOpen** ()
  Get open circuit detection result.
- UeiDaqAPI bool **IsBiasEnabled** ()
  Get bias resistor state.
- UeiDaqAPI void **EnableBias** (bool enable)
  Set bias resistor state.
- UeiDaqAPI bool **IsAutoZeroEnabled** ()
  Get auto-zero state.
- UeiDaqAPI void **EnableAutoZero** (bool enable)
  Set auto zero state.
- UeiDaqAPI int **GetMovingAverageWindowSize** ()
  Get moving average window size.
- UeiDaqAPI void **SetMovingAverageWindowSize** (int windowSize)
  Set moving average window size.
- UeiDaqAPI **tUeiAIChannelMuxPos GetMuxPos** ()
  Get mux position mode.
- UeiDaqAPI void **SetMuxPos** (**tUeiAIChannelMuxPos** muxPos)
  Set mux position mode.
- UeiDaqAPI bool **IsVoltageDividerEnabled** ()
  Get state of input voltage divider.
- UeiDaqAPI void **EnableVoltageDivider** (bool enable)
  Set state of input voltage divider.

2.41.1 Detailed Description

Manages settings for each voltage Analog input channel

2.41.2 Member Function Documentation

2.41.2.1 UeiDaqAPI f64 UeiDaq::CUeiAIChannel::GetMinimum ()

Get the minimum expected value for the signal connected to the channel.

Returns:

the minimum value.

See also:

SetMinimum (p. 53)

2.41.2.2 UeiDaqAPI void UeiDaq::CUeiAIChannel::SetMinimum (f64 *minimum*)

Set the minimum expected value for the signal connected to the channel.

Parameters:

minimum the minimum value.

See also:

GetMinimum (p. 52)

2.41.2.3 UeiDaqAPI f64 UeiDaq::CUeiAIChannel::GetMaximum ()

Get the maximum expected value for the signal connected to the channel.

Returns:

the maximum value.

See also:

SetMaximum (p. 53)

2.41.2.4 UeiDaqAPI void UeiDaq::CUeiAIChannel::SetMaximum (f64 *maximum*)

Set the maximum expected value for the signal connected to the channel.

Parameters:

maximum the maximum value.

See also:

GetMaximum (p. 53)

2.41.2.5 UeiDaqAPI tUeiAIChannelInputMode UeiDaq::CUeiAIChannel::GetInputMode ()

Get the Input mode of the channel, possible values are "differential" or "single-ended".

Returns:

the input mode.

See also:

SetInputMode (p. 54)

2.41.2.6 UeiDaqAPI void UeiDaq::CUeiAIChannel::SetInputMode (tUeiAIChannelInputMode *mode*)

Set the Input mode of the channel, possible values are "differential" or "single-ended".

Parameters:

mode the input mode.

See also:

GetInputMode (p. 53)

2.41.2.7 UeiDaqAPI f64 UeiDaq::CUeiAIChannel::GetGain ()

Get the gain that best fits the requested input range

Returns:

the gain

See also:

SetGain (p. 54)

2.41.2.8 UeiDaqAPI void UeiDaq::CUeiAIChannel::SetGain (f64 *gain*)

Set the gain

Parameters:

gain the new gain

See also:

GetGain (p. 54)

2.41.2.9 UeiDaqAPI CUeiCustomScale* UeiDaq::CUeiAIChannel::GetCustomScale ()

Get the custom scale applied to scaled measurements before they are returned to caller

Returns:

Pointer to the current custom scale object

See also:

SetCustomScale (p. 55)

2.41.2.10 UeiDaqAPI void UeiDaq::CUeiAIChannel::SetCustomScale (CUeiCustomScale * *pScale*)

Set the custom scale to apply to scaled measurements before they are returned to caller

Parameters:

pScale Pointer to the custom scale object

See also:

GetCustomScale (p. 54)

2.41.2.11 UeiDaqAPI bool UeiDaq::CUeiAIChannel::IsOpenCircuitTestEnabled ()

Get status of open circuit detection

Returns:

true if open circuit detection is enabled, false otherwise

See also:

EnableOpenCircuitTest (p. 55)

2.41.2.12 UeiDaqAPI void UeiDaq::CUeiAIChannel::EnableOpenCircuitTest (bool *enable*)

Enable or disable open circuit detection

Parameters:

enable true to enable open circuit detection

See also:

IsOpenCircuitTestEnabled (p. 55)

2.41.2.13 UeiDaqAPI bool UeiDaq::CUeiAIChannel::IsCircuitOpen ()

Get result of open circuit detection

Returns:

true if circuit is open, false otherwise

See also:

EnableOpenCircuitTest (p. 55)

2.41.2.14 UeiDaqAPI bool UeiDaq::CUeiAIChannel::IsBiasEnabled ()

Bias resistors connects the negative input to ground and positive input to a voltage source. This is useful to measure temperature from un-grounded thermocouple. Disable bias to measure from grounded thermocouples

Returns:

the bias resistor state

2.41.2.15 UeiDaqAPI void UeiDaq::CUeiAIChannel::EnableBias (bool *enable*)

Bias resistors connects the negative input to ground and positive input to a voltage source. This is useful to measure temperature from un-grounded thermocouple. Disable bias to measure from grounded thermocouples

Parameters:

enable the new bias resistor state

2.41.2.16 UeiDaqAPI bool UeiDaq::CUeiAIChannel::IsAutoZeroEnabled ()

Some analog input devices come with a special channel to measure ground offset. auto-zero subtract the ground voltage measurement from the input voltage measurement.

Returns:

the auto-zero state

2.41.2.17 UeiDaqAPI void UeiDaq::CUeiAIChannel::EnableAutoZero (bool *enable*)

Some analog input devices come with a special channel to measure ground offset. auto-zero subtract the ground voltage measurement from the input voltage measurement.

Parameters:

enable the new auto-zero state

2.41.2.18 UeiDaqAPI int UeiDaq::CUeiAIChannel::GetMovingAverageWindowSize ()

Get the moving average windows size for devices that have this capability The device needs to be capable of averaging measurement in hardware

Returns:

the current moving average window size

2.41.2.19 UeiDaqAPI void UeiDaq::CUeiAIChannel::SetMovingAverageWindowSize (int *windowSize*)

Set the moving average windows size for devices that have this capability This is the number of data points used to calculate the average The device needs to be capable of averaging measurements in hardware

Parameters:

windowSize the new moving average window size

2.41.2.20 UeiDaqAPI tUeiAIChannelMuxPos UeiDaq::CUeiAIChannel::GetMuxPos ()

Get the mux position mode for devices with self-test capability return the current mux position

2.41.2.21 UeiDaqAPI void UeiDaq::CUeiAIChannel::SetMuxPos (tUeiAIChannelMuxPos *muxPos*)

Set the mux position mode for devices with self-test capability

Parameters:

muxPos the new mux position

2.41.2.22 UeiDaqAPI bool UeiDaq::CUeiAIChannel::IsVoltageDividerEnabled ()

check whether input voltage divider is enabled return the current mux position

2.41.2.23 UeiDaqAPI void UeiDaq::CUeiAIChannel::EnableVoltageDivider (bool *enable*)

Enable or disable the input voltage divider

Parameters:

enable true to enable voltage divider, false otherwise

The documentation for this class was generated from the following file:

- UeiChannel.h

2.42 UeiDaq::CUeiAICurrentChannel Class Reference

Manages settings for each current input channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiAIChannel**.

Public Member Functions

- UeiDaqAPI **CUeiAICurrentChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiAICurrentChannel** ()
  Destructor.
- UeiDaqAPI **tUeiFeatureEnable IsCircuitBreakerEnabled** ()
  Determines whether the CB is currently protecting the channel.
- UeiDaqAPI void **EnableCircuitBreaker** (**tUeiFeatureEnable** enable)
  Enable or Disable channel protection.
- UeiDaqAPI double **GetCircuitBreakerHighLimit** ()
- UeiDaqAPI void **SetCircuitBreakerHighLimit** (double highLimit)

2.42.1 Detailed Description

Manages settings for each current input channel

2.42.2 Member Function Documentation

2.42.2.1 UeiDaqAPI tUeiFeatureEnable UeiDaq::CUeiAICurrentChannel::IsCircuitBreakerEnabled ()

Return true is circuit breaker for this channels is enabled and false otherwise

Returns:

Circuit breaker state

2.42.2.2 UeiDaqAPI void UeiDaq::CUeiAICurrentChannel::EnableCircuitBreaker (tUeiFeatureEnable *enable*)

Enable or disable circuit breaker on this channel. when enabled a circuit breaker monitors the current flowing through the channel and opens the circuit if the current goes out of pre-set limits.

Parameters:

enable True to turn-on protection, false to turn it off

2.42.2.3 UeiDaqAPI double UeiDaq::CUeiAICurrentChannel::GetCircuitBreakerHighLimit ()

Get Circuit breaker high limit

Current input channels are equipped with a circuit-breaker that monitors current and open the circuit if the monitored current is higher than the high limit.

Returns:

the high current limit value.

2.42.2.4 UeiDaqAPI void UeiDaq::CUeiAICurrentChannel::SetCircuitBreakerHighLimit (double *highLimit*)

Set Circuit breaker high limit

Current input channels are equipped with a circuit-breaker that monitors current and open the circuit if the monitored current is higher than the high limit.

Parameters:

highLimit the new high current limit

The documentation for this class was generated from the following file:

- UeiChannel.h

2.43 UeiDaq::CUeiAIVExChannel Class Reference

Manages settings for each voltage with excitation input channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiAIChannel**.

Public Member Functions

- UeiDaqAPI **CUeiAIVExChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiAIVExChannel** ()
  Destructor.
- UeiDaqAPI **tUeiSensorBridgeType GetBridgeType** ()
  Get the bridge type.
- UeiDaqAPI void **SetBridgeType** (**tUeiSensorBridgeType** type)
  Set the bridge type.
- UeiDaqAPI double **GetExcitationVoltage** ()
  Get the excitation voltage.
- UeiDaqAPI void **SetExcitationVoltage** (double vex)
  Set the excitation voltage.
- UeiDaqAPI double **GetMeasuredExcitationVoltage** ()
  Get the measured excitation voltage.
- UeiDaqAPI void **SetExcitationFrequency** (double fex)
  Set the excitation frequency.
- UeiDaqAPI double **GetExcitationFrequency** ()
  Get the excitation frequency.
- UeiDaqAPI double **GetActualExcitationFrequency** ()
  Get the actual excitation frequency.
- UeiDaqAPI bool **GetScalingWithExcitation** ()
  Get the scaling mode.
- UeiDaqAPI void **SetScalingWithExcitation** (bool scaleWithExcitation)
  Set the scaling mode.
- UeiDaqAPI bool **IsShuntCalibrationEnabled** ()
  Get the shunt calibration resistor status.
- UeiDaqAPI void **EnableShuntCalibration** (bool engageShuntCal)

Control the shunt calibration resistor.

- UeiDaqAPI **tUeiWheatstoneBridgeBranch GetShuntLocation** ()
  Get the shunt resistor location.
- UeiDaqAPI void **SetShuntLocation** (**tUeiWheatstoneBridgeBranch** shuntedBranch)
  Set the shunt resistor location.
- UeiDaqAPI double **GetShuntResistance** ()
  Get the shunt resistance.
- UeiDaqAPI void **SetShuntResistance** (double resistance)
  Set the shunt resistance.
- UeiDaqAPI double **GetActualShuntResistance** ()
  Get the actual shunt resistance.
- UeiDaqAPI double **GetGainAdjustmentFactor** ()
  Get the gain adjustment factor.
- UeiDaqAPI void **SetGainAdjustmentFactor** (double gaf)
  Set the gain adjustment factor.
- UeiDaqAPI void **SetWiringScheme** (**tUeiWiringScheme** wiring)
  Set the wiring scheme.
- UeiDaqAPI **tUeiWiringScheme GetWiringScheme** ()
  Get the wiring scheme.
- UeiDaqAPI bool **IsOffsetNullingEnabled** ()
  Get the offset nulling circuitry status.
- UeiDaqAPI void **EnableOffsetNulling** (bool enableNulling)
  Control the offset nulling circuitry.
- UeiDaqAPI bool **IsAutoOffsetNullingEnabled** ()
  Get the automatic offset nulling status.
- UeiDaqAPI void **EnableAutoOffsetNulling** (bool enableAutoNulling)
  Control the automatic offset nulling.
- UeiDaqAPI double **GetOffsetNullingSetting** ()
  Get the offset nulling setting.
- UeiDaqAPI void **SetOffsetNullingSetting** (double setting)
  Set the offset nulling setting.
- UeiDaqAPI bool **IsAutoBridgeCompletionEnabled** ()
  Get the automatic bridge completion status.

- UeiDaqAPI void **EnableAutoBridgeCompletion** (bool enableAutoBridgeCompletion)
  Control the automatic bridge completion.
- UeiDaqAPI double **GetBridgeCompletionSetting** ()
  Get the bridge completion setting.
- UeiDaqAPI void **SetBridgeCompletionSetting** (double setting)
  Set the offset nulling setting.

2.43.1 Detailed Description

Manages settings for each voltage with excitation input channel for sensors that require excitation such as load cells and pressure sensors.

2.43.2 Member Function Documentation

2.43.2.1 UeiDaqAPI tUeiSensorBridgeType UeiDaq::CUeiAIVExChannel::GetBridgeType ()

Get the bridge type configured for the sensor connected to this channel.

Returns:

the bridge type.

See also:

SetBridgeType (p. 62)

2.43.2.2 UeiDaqAPI void UeiDaq::CUeiAIVExChannel::SetBridgeType (tUeiSensorBridgeType *type*)

Set the bridge type for the sensor connected to this channel.

Parameters:

type The bridge type

See also:

GetBridgeType (p. 62)

2.43.2.3 UeiDaqAPI double UeiDaq::CUeiAIVExChannel::GetExcitationVoltage ()

Get the excitation voltage configured for the channel.

Returns:

The excitation voltage.

See also:

SetExcitationVoltage (p. 63)

2.43.2.4 UeiDaqAPI void UeiDaq::CUeiAIVExChannel::SetExcitationVoltage (double *vex*)

Set the excitation voltage for this channel.

Parameters:

vex The excitation voltage

See also:

GetExcitationVoltage (p. 62)

2.43.2.5 UeiDaqAPI double UeiDaq::CUeiAIVExChannel::GetMeasuredExcitationVoltage ()

Get the actual excitation voltage measured on this channel.

Returns:

The actual measured excitation voltage.

See also:

GetExcitationVoltage (p. 62) **SetExcitationVoltage** (p. 63)

2.43.2.6 UeiDaqAPI void UeiDaq::CUeiAIVExChannel::SetExcitationFrequency (double *fex*)

Set the excitation frequency for this channel. Set frequency to 0.0 for DC excitation.

Parameters:

fex The excitation frequency

See also:

GetExcitationFrequency (p. 63)

2.43.2.7 UeiDaqAPI double UeiDaq::CUeiAIVExChannel::GetExcitationFrequency ()

Get the excitation frequency configured for the channel.

Returns:

The excitation frequency.

See also:

SetExcitationFrequency (p. 63)

2.43.2.8 UeiDaqAPI double UeiDaq::CUeiAIVExChannel::GetActualExcitationFrequency ()

Get the actual excitation frequency generated for this channel.

Returns:

The actual excitation frequency.

See also:

GetExcitationFrequency (p. 63) **SetExcitationFrequency** (p. 63)

2.43.2.9 UeiDaqAPI bool UeiDaq::CUeiAIVExChannel::GetScalingWithExcitation ()

Determines if acquired data will be divided by the excitation before being returned.

Returns:

the scaling mode .

See also:

SetScalingWithExcitation (p. 64)

2.43.2.10 UeiDaqAPI void UeiDaq::CUeiAIVExChannel::SetScalingWithExcitation (bool *scaleWithExcitation*)

Specifies whether the acquired data will be divided by the excitation voltage before it is returned.

Parameters:

scaleWithExcitation The scaling mode

See also:

GetScalingWithExcitation (p. 64)

2.43.2.11 UeiDaqAPI bool UeiDaq::CUeiAIVExChannel::IsShuntCalibrationEnabled ()

Determines whether the shunt calibration resistor is connected.

Returns:

true is the shunt resistor is connected, false otherwise.

See also:

EnableShuntCalibration (p. 65)

2.43.2.12 UeiDaqAPI void UeiDaq::CUeiAIVExChannel::EnableShuntCalibration (bool *engageShuntCal*)

Connect or disconnect the shunt calibration resistor.

Parameters:

engageShuntCal true to enable the shunt resistor, false to disable it

See also:

IsShuntCalibrationEnabled (p. 64)

2.43.2.13 UeiDaqAPI tUeiWheatstoneBridgeBranch UeiDaq::CUeiAIVExChannel::GetShuntLocation ()

Get the branch of the wheatstone bridge on which the shunt resistor is connected.

Returns:

The shunted bridge branch.

See also:

SetShuntLocation (p. 65)

2.43.2.14 UeiDaqAPI void UeiDaq::CUeiAIVExChannel::SetShuntLocation (tUeiWheatstoneBridgeBranch *shuntedBranch*)

Specifies the wheatstone bridge branch to shunt.

Parameters:

shuntedBranch The bridge branch to shunt.

See also:

GetShuntLocation (p. 65)

2.43.2.15 UeiDaqAPI double UeiDaq::CUeiAIVExChannel::GetShuntResistance ()

Get the resistance programmed to the shunt resistor in Ohms. NOTE: This is not the actual resistance used to perform the calibration. The resistance of other components must be accounted for, use **GetActualShuntResistance()** (p. 66) to get the shunt resistance to use to calculate the gain adjustment factor.

Returns:

The programmed resistance.

See also:

SetShuntResistance (p. 66) **GetActualShuntResistance** (p. 66)

2.43.2.16 UeiDaqAPI void UeiDaq::CUeiAIVExChannel::SetShuntResistance (double *resistance*)

Specifies the resistance to program to the shunt resistor in Ohms. NOTE: Don't use this value to calculate the gain adjustment factor, Use the value returned by GetActualShuntResitance() which takes into account the resistance of other non-programmable parts in the shunt circuit.

Parameters:

resistance The resistance to program on the shunt resistor.

See also:

GetShuntResistance (p. 65) **GetActualShuntResistance** (p. 66)

2.43.2.17 UeiDaqAPI double UeiDaq::CUeiAIVExChannel::GetActualShuntResistance ()

Get the actual shunt resistance in Ohms. This resistance includes the programmed shunt resistance plus the resistance of other parts in the shunt circuitry.

Returns:

The programmed resistance.

See also:

SetShuntResistance (p. 66) **GetShuntResistance** (p. 65)

2.43.2.18 UeiDaqAPI double UeiDaq::CUeiAIVExChannel::GetGainAdjustmentFactor ()

Get the gain adjustment factor.

Returns:

The gain adjustment factor.

See also:

SetGainAdjustmentFactor (p. 66)

2.43.2.19 UeiDaqAPI void UeiDaq::CUeiAIVExChannel::SetGainAdjustmentFactor (double *gaf*)

Setting the gain adjustment factor is part of the shunt calibration procedure: You must first program the shunt resistance, engage the the shunt resistor and measure some values. You can then calculate the gain adjustment factor with the formula: gaf = Vex∗(Rg/(4∗Rs + 2∗Rg))/(Vout-Shunted - Vout) Vex: excitation voltage Vout: measured voltage with shunt disabled VoutShunted: measured voltage with shunt enabled Rg: Resistance of the strain gages. Rs: Resistance of the shunt.

Once the gain adjustment factor is set, the framework automatically uses it to scale measurements to engineering unit.

Parameters:

gaf The new gain adjustment factor.

See also:

GetGainAdjustmentFactor (p. 66)

2.43.2.20 UeiDaqAPI void UeiDaq::CUeiAIVExChannel::SetWiringScheme (tUeiWiringScheme *wiring*)

Specifies the wiring scheme used to connect the strain gage or load cell to the channel.

Parameters:

wiring The new wiring scheme.

See also:

GetWiringScheme (p. 67)

2.43.2.21 UeiDaqAPI tUeiWiringScheme UeiDaq::CUeiAIVExChannel::GetWiringScheme ()

Get the current wiring scheme used for this channel.

Returns:

The current wiring scheme.

See also:

SetWiringScheme (p. 67)

2.43.2.22 UeiDaqAPI bool UeiDaq::CUeiAIVExChannel::IsOffsetNullingEnabled ()

Determines whether the offset nulling circuitry is enabled. With Offset nulling enabled, a nulling circuit adds an adjustable DC voltage to the output of the amplifier making sure that the bridge output is 0V when no strain is applied.

Returns:

true if offset nulling is enabled, false otherwise.

See also:

EnableOffsetNulling (p. 68)

2.43.2.23 UeiDaqAPI void UeiDaq::CUeiAIVExChannel::EnableOffsetNulling (bool *enableNulling*)

Enables or disables offset nulling circuitry. With Offset nulling enabled, a nulling circuit adds an adjustable DC voltage to the output of the amplifier making sure that the bridge output is 0V when no strain is applied.

Parameters:

enableNulling true to enable offset nulling, false to disable it

See also:

IsOffsetNullingEnabled (p. 67)

2.43.2.24 UeiDaqAPI bool UeiDaq::CUeiAIVExChannel::IsAutoOffsetNullingEnabled ()

Determines whether the offset nulling is automatically set.

Returns:

true if automatic offset nulling is enabled, false otherwise.

See also:

EnableAutoOffsetNulling (p. 68)

2.43.2.25 UeiDaqAPI void UeiDaq::CUeiAIVExChannel::EnableAutoOffsetNulling (bool *enableAutoNulling*)

Enables or disables automatic offset nulling.

Parameters:

enableAutoNulling true to enable automatic offset nulling, false to disable it

See also:

IsAutoOffsetNullingEnabled (p. 68)

2.43.2.26 UeiDaqAPI double UeiDaq::CUeiAIVExChannel::GetOffsetNullingSetting ()

Get the offset nulling setting used to program the nulling circuitry. Set it to 0.0 to automatically perform offset nulling next time the session is started. Make sure no strain is applied on the bridge before nulling the offset.

Returns:

The current offset nulling setting.

See also:

SetOffsetNullingSetting (p. 69)

2.43.2.27 UeiDaqAPI void UeiDaq::CUeiAIVExChannel::SetOffsetNullingSetting (double *setting*)

Set the offset nulling setting used to program the nulling circuitry. Set it to 0.0 to automatically perform offset nulling next time the session is started. Make sure no strain is applied on the bridge before nulling the offset.

Parameters:

setting The setting value used to program the offset nulling circuitry.

See also:

GetOffsetNullingSetting (p. 68)

2.43.2.28 UeiDaqAPI bool UeiDaq::CUeiAIVExChannel::IsAutoBridgeCompletionEnabled ()

Determines whether the bridge completion is automatically set.

Returns:

true if automatic bridge completion is enabled, false otherwise.

See also:

EnableAutoBridgeCompletion (p. 69)

2.43.2.29 UeiDaqAPI void UeiDaq::CUeiAIVExChannel::EnableAutoBridgeCompletion (bool *enableAutoBridgeCompletion*)

Enables or disables automatic bridge completion.

Parameters:

enableAutoBridgeCompletion true to enable automatic bridge completion, false to disable it

See also:

IsAutoBridgeCompletionEnabled (p. 69)

2.43.2.30 UeiDaqAPI double UeiDaq::CUeiAIVExChannel::GetBridgeCompletionSetting ()

Get the bridge completion setting used to program the bridge completion circuitry. Set it to 0.0 to automatically perform bridge completion next time the session is started. Make sure no strain is applied on the bridge.

Returns:

The current bridge completion setting.

See also:

SetBridgeCompletionSetting (p. 70)

2.43.2.31 UeiDaqAPI void UeiDaq::CUeiAIVExChannel::SetBridgeCompletionSetting (double *setting*)

Set the bridge completion setting used to program the bridge completion circuitry. Set it to 0.0 to automatically perform bridge completion next time the session is started. Make sure no strain is applied on the bridge.

Parameters:

setting The setting value used to program the bridge completion circuitry.

See also:

GetBridgeCompletionSetting (p. 69)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.44 UeiDaq::CUeiAnalogCalibratedRawReader Class Reference

Analog Calibrated Raw Reader class.

`#include <UeiReader.h>`

Public Member Functions

- UeiDaqAPI **CUeiAnalogCalibratedRawReader** (**CUeiDataStream** *pDataStream)
  Constructor.
- virtual UeiDaqAPI ~**CUeiAnalogCalibratedRawReader** ()
  Destructor.
- UeiDaqAPI void **ReadSingleScan** (**uInt16** *pBuffer)
  Read a 16 bits wide scan.
- UeiDaqAPI void **ReadSingleScan** (**uInt32** *pBuffer)
  Read a 32 bits wide scan.
- UeiDaqAPI void **ReadMultipleScans** (**Int32** numScans, **uInt16** *pBuffer)
  Read multiple 16 bits wide scans.
- UeiDaqAPI void **ReadMultipleScans** (**Int32** numScans, **uInt32** *pBuffer)
  Read multiple 32 bits wide scans.
- UeiDaqAPI void **ReadSingleScanAsync** (**uInt16** *pBuffer)
  Read a 16 bits wide scan asynchronously.
- UeiDaqAPI void **ReadSingleScanAsync** (**uInt32** *pBuffer)
  Read a 32 bits wide scan asynchronously.
- UeiDaqAPI void **ReadMultipleScansAsync** (**Int32** numScans, **uInt16** *pBuffer)
  Read multiple 16 bits wide scans asynchronously.
- UeiDaqAPI void **ReadMultipleScansAsync** (**Int32** numScans, **uInt32** *pBuffer)
  Read multiple 32 bits wide scans asynchronously.
- UeiDaqAPI void **AddEventListener** (**IUeiEventListener** *pListener)
  Add an asynchronous listener.

2.44.1 Detailed Description

Class that handles reading calibrated raw data of a stream coming from an analog input

2.44.2 Constructor & Destructor Documentation

2.44.2.1 UeiDaqAPI UeiDaq::CUeiAnalogCalibratedRawReader::CUeiAnalogCalibrated-RawReader (CUeiDataStream * *pDataStream*)

Parameters:

pDataStream represents the source of data to read

2.44.3 Member Function Documentation

2.44.3.1 UeiDaqAPI void UeiDaq::CUeiAnalogCalibratedRawReader::ReadSingleScan (uInt16 * *pBuffer*)

Read only one scan from the input stream

Parameters:

pBuffer destination buffer

2.44.3.2 UeiDaqAPI void UeiDaq::CUeiAnalogCalibratedRawReader::ReadSingleScan (uInt32 * *pBuffer*)

Read only one scan from the input stream

Parameters:

pBuffer destination buffer

2.44.3.3 UeiDaqAPI void UeiDaq::CUeiAnalogCalibratedRawReader::ReadMultipleScans (Int32 *numScans*, uInt16 * *pBuffer*)

Read several scans from the input stream

Parameters:

numScans number of scans to read

pBuffer destination buffer

2.44.3.4 UeiDaqAPI void UeiDaq::CUeiAnalogCalibratedRawReader::ReadMultipleScans (Int32 *numScans*, uInt32 * *pBuffer*)

Read several scans from the input stream

Parameters:

numScans number of scans to read

pBuffer destination buffer

2.44.3.5 UeiDaqAPI void UeiDaq::CUeiAnalogCalibratedRawReader::ReadSingleScanAsync (uInt16 * *pBuffer*)

Read only one scan asynchronously from the input stream. The destination buffer contains valid data once the listener is called.

Parameters:

pBuffer destination buffer

2.44.3.6 UeiDaqAPI void UeiDaq::CUeiAnalogCalibratedRawReader::ReadSingleScanAsync (uInt32 * *pBuffer*)

Read only one scan asynchronously from the input stream. The destination buffer contains valid data once the listener is called.

Parameters:

pBuffer destination buffer

2.44.3.7 UeiDaqAPI void UeiDaq::CUeiAnalogCalibratedRawReader::ReadMultipleScansAsync (Int32 *numScans*, uInt16 * *pBuffer*)

Read several scans asynchronously from the input stream. The destination buffer contains valid data once the listener is called.

Parameters:

numScans number of scans to read
pBuffer destination buffer

2.44.3.8 UeiDaqAPI void UeiDaq::CUeiAnalogCalibratedRawReader::ReadMultipleScansAsync (Int32 *numScans*, uInt32 * *pBuffer*)

Read several scans asynchronously from the input stream. The destination buffer contains valid data once the listener is called.

Parameters:

numScans number of scans to read
pBuffer destination buffer

2.44.3.9 UeiDaqAPI void UeiDaq::CUeiAnalogCalibratedRawReader::AddEventListener (IUeiEventListener * *pListener*)

Subscribe a listener to receive asynchronous events

Parameters:

pListener pointer to a class that implements **IUeiEventListener** (p. 549) interface

The documentation for this class was generated from the following file:

- UeiReader.h

2.45 UeiDaq::CUeiAnalogRawReader Class Reference

Analog Raw Reader class.

`#include <UeiReader.h>`

Public Member Functions

- UeiDaqAPI **CUeiAnalogRawReader** (**CUeiDataStream** *pDataStream)
  Constructor.
- virtual UeiDaqAPI ~**CUeiAnalogRawReader** ()
  Destructor.
- UeiDaqAPI void **ReadSingleScan** (**uInt16** *pBuffer)
  Read a 16 bits wide scan.
- UeiDaqAPI void **ReadSingleScan** (**uInt32** *pBuffer)
  Read a 32 bits wide scan.
- UeiDaqAPI void **ReadMultipleScans** (**Int32** numScans, **uInt16** *pBuffer)
  Read multiple 16 bits wide scans.
- UeiDaqAPI void **ReadMultipleScans** (**Int32** numScans, **uInt32** *pBuffer)
  Read multiple 32 bits wide scans.
- UeiDaqAPI void **ReadSingleScanAsync** (**uInt16** *pBuffer)
  Read a 16 bits wide scan asynchronously.
- UeiDaqAPI void **ReadSingleScanAsync** (**uInt32** *pBuffer)
  Read a 32 bits wide scan asynchronously.
- UeiDaqAPI void **ReadMultipleScansAsync** (**Int32** numScans, **uInt16** *pBuffer)
  Read multiple 16 bits wide scans asynchronously.
- UeiDaqAPI void **ReadMultipleScansAsync** (**Int32** numScans, **uInt32** *pBuffer)
  Read multiple 32 bits wide scans asynchronously.
- UeiDaqAPI void **AddEventListener** (**IUeiEventListener** *pListener)
  Add an asynchronous listener.

2.45.1 Detailed Description

Class that handles reading raw data of a stream coming from an analog input

2.45.2 Constructor & Destructor Documentation

2.45.2.1 UeiDaqAPI UeiDaq::CUeiAnalogRawReader::CUeiAnalogRawReader (CUeiDataStream * *pDataStream*)

Parameters:

pDataStream represents the source of data to read

2.45.3 Member Function Documentation

2.45.3.1 UeiDaqAPI void UeiDaq::CUeiAnalogRawReader::ReadSingleScan (uInt16 * *pBuffer*)

Read only one scan from the input stream

Parameters:

pBuffer destination buffer

2.45.3.2 UeiDaqAPI void UeiDaq::CUeiAnalogRawReader::ReadSingleScan (uInt32 * *pBuffer*)

Read only one scan from the input stream

Parameters:

pBuffer destination buffer

2.45.3.3 UeiDaqAPI void UeiDaq::CUeiAnalogRawReader::ReadMultipleScans (Int32 *numScans*, uInt16 * *pBuffer*)

Read several scans from the input stream

Parameters:

numScans number of scans to read

pBuffer destination buffer

2.45.3.4 UeiDaqAPI void UeiDaq::CUeiAnalogRawReader::ReadMultipleScans (Int32 *numScans*, uInt32 * *pBuffer*)

Read several scans from the input stream

Parameters:

numScans number of scans to read

pBuffer destination buffer

2.45.3.5 UeiDaqAPI void UeiDaq::CUeiAnalogRawReader::ReadSingleScanAsync (uInt16 * *pBuffer*)

Read only one scan asynchronously from the input stream. The destination buffer contains valid data once the listener is called.

Parameters:

pBuffer destination buffer

2.45.3.6 UeiDaqAPI void UeiDaq::CUeiAnalogRawReader::ReadSingleScanAsync (uInt32 * *pBuffer*)

Read only one scan asynchronously from the input stream. The destination buffer contains valid data once the listener is called.

Parameters:

pBuffer destination buffer

2.45.3.7 UeiDaqAPI void UeiDaq::CUeiAnalogRawReader::ReadMultipleScansAsync (Int32 *numScans*, uInt16 * *pBuffer*)

Read several scans asynchronously from the input stream. The destination buffer contains valid data once the listener is called.

Parameters:

numScans number of scans to read
pBuffer destination buffer

2.45.3.8 UeiDaqAPI void UeiDaq::CUeiAnalogRawReader::ReadMultipleScansAsync (Int32 *numScans*, uInt32 * *pBuffer*)

Read several scans asynchronously from the input stream. The destination buffer contains valid data once the listener is called.

Parameters:

numScans number of scans to read
pBuffer destination buffer

2.45.3.9 UeiDaqAPI void UeiDaq::CUeiAnalogRawReader::AddEventListener (IUeiEventListener * *pListener*)

Subscribe a listener to receive asynchronous events

Parameters:

pListener pointer to a class that implements **IUeiEventListener** (p. 549) interface

The documentation for this class was generated from the following file:

- UeiReader.h

2.46 UeiDaq::CUeiAnalogRawWriter Class Reference

Analog Raw Writer class.

`#include <UeiWriter.h>`

Public Member Functions

- UeiDaqAPI **CUeiAnalogRawWriter** (**CUeiDataStream** *pDataStream)
  Constructor.
- virtual UeiDaqAPI ~**CUeiAnalogRawWriter** ()
  Destructor.
- UeiDaqAPI void **WriteSingleScan** (**uInt16** *pBuffer)
  Write a 16 bits wide scan.
- UeiDaqAPI void **WriteSingleScan** (**uInt32** *pBuffer)
  Write a 32 bits wide scan.
- UeiDaqAPI void **WriteMultipleScans** (**Int32** numScans, **uInt16** *pBuffer)
  Write multiple 16 bits wide scans.
- UeiDaqAPI void **WriteMultipleScans** (**Int32** numScans, **uInt32** *pBuffer)
  Write multiple 32 bits wide scans.
- UeiDaqAPI void **WriteSingleScanAsync** (**uInt16** *pBuffer)
  Write a 16 bits wide scan asynchronously.
- UeiDaqAPI void **WriteSingleScanAsync** (**uInt32** *pBuffer)
  Write a 32 bits wide scan asynchronously.
- UeiDaqAPI void **WriteMultipleScansAsync** (**Int32** numScans, **uInt16** *pBuffer)
  Write multiple 16 bits wide scans asynchronously.
- UeiDaqAPI void **WriteMultipleScansAsync** (**Int32** numScans, **uInt32** *pBuffer)
  Write multiple 32 bits wide scans asynchronously.
- UeiDaqAPI void **AddEventListener** (**IUeiEventListener** *pListener)
  Add an asynchronous listener.

2.46.1 Detailed Description

Class that handles writing raw data to a stream going to an analog output

2.46.2 Constructor & Destructor Documentation

2.46.2.1 UeiDaqAPI UeiDaq::CUeiAnalogRawWriter::CUeiAnalogRawWriter (CUeiDataStream * *pDataStream*)

Parameters:

pDataStream represents the destination where to write data

2.46.3 Member Function Documentation

2.46.3.1 UeiDaqAPI void UeiDaq::CUeiAnalogRawWriter::WriteSingleScan (uInt16 * *pBuffer*)

Write only one scan to the output stream

Parameters:

pBuffer source buffer

2.46.3.2 UeiDaqAPI void UeiDaq::CUeiAnalogRawWriter::WriteSingleScan (uInt32 * *pBuffer*)

Write only one scan to the output stream

Parameters:

pBuffer source buffer

2.46.3.3 UeiDaqAPI void UeiDaq::CUeiAnalogRawWriter::WriteMultipleScans (Int32 *numScans*, uInt16 * *pBuffer*)

Write several scans to the output stream

Parameters:

numScans number of scans to write

pBuffer source buffer

2.46.3.4 UeiDaqAPI void UeiDaq::CUeiAnalogRawWriter::WriteMultipleScans (Int32 *numScans*, uInt32 * *pBuffer*)

Write several scans to the output stream

Parameters:

numScans number of scans to write

pBuffer source buffer

2.46.3.5 UeiDaqAPI void UeiDaq::CUeiAnalogRawWriter::WriteSingleScanAsync (uInt16 * *pBuffer*)

Write only one scan asynchronously to the output stream. The source buffer can be reused once the listener is called.

Parameters:

pBuffer source buffer

2.46.3.6 UeiDaqAPI void UeiDaq::CUeiAnalogRawWriter::WriteSingleScanAsync (uInt32 * *pBuffer*)

Write only one scan asynchronously to the output stream. The source buffer can be reused once the listener is called.

Parameters:

pBuffer source buffer

2.46.3.7 UeiDaqAPI void UeiDaq::CUeiAnalogRawWriter::WriteMultipleScansAsync (Int32 *numScans*, uInt16 * *pBuffer*)

Write several scans asynchronously to the output stream. The source buffer can be reused once the listener is called.

Parameters:

numScans number of scans to write
pBuffer source buffer

2.46.3.8 UeiDaqAPI void UeiDaq::CUeiAnalogRawWriter::WriteMultipleScansAsync (Int32 *numScans*, uInt32 * *pBuffer*)

Write several scans asynchronously to the output stream. The source buffer can be reused once the listener is called.

Parameters:

numScans number of scans to write
pBuffer source buffer

2.46.3.9 UeiDaqAPI void UeiDaq::CUeiAnalogRawWriter::AddEventListener (IUeiEventListener * *pListener*)

Subscribe a listener to receive asynchronous events

Parameters:

pListener pointer to a class that implements **IUeiEventListener** (p. 549) interface

The documentation for this class was generated from the following file:

- UeiWriter.h

2.47 UeiDaq::CUeiAnalogScaledReader Class Reference

Analog Scaled Reader class.

`#include <UeiReader.h>`

Public Member Functions

- UeiDaqAPI **CUeiAnalogScaledReader** (**CUeiDataStream** *pDataStream)
  Constructor.
- virtual UeiDaqAPI ~**CUeiAnalogScaledReader** ()
  Destructor.
- UeiDaqAPI void **ReadSingleScan** (**f64** *pBuffer)
  Read a scan.
- UeiDaqAPI void **ReadMultipleScans** (**Int32** numScans, **f64** *pBuffer)
  Read multiple scans.
- UeiDaqAPI void **ReadSingleScanAsync** (**f64** *pBuffer)
  Read a scan asynchronously.
- UeiDaqAPI void **ReadMultipleScansAsync** (**Int32** numScans, **f64** *pBuffer)
  Read multiple scans asynchronously.
- UeiDaqAPI void **AddEventListener** (**IUeiEventListener** *pListener)
  Add an asynchronous listener.

2.47.1 Detailed Description

Class that handles reading scaled data of a stream coming from an analog input

Examples:

AnalogInBuffered.cpp, **AnalogInBufferedAsync.cpp**, **AnalogInOneShot_Trig_Ex.cpp**, and **AnalogInSingle.cpp**.

2.47.2 Constructor & Destructor Documentation

2.47.2.1 UeiDaqAPI UeiDaq::CUeiAnalogScaledReader::CUeiAnalogScaledReader (CUeiDataStream * *pDataStream*)

Parameters:

pDataStream represents the source of data to read

2.47.3 Member Function Documentation

2.47.3.1 UeiDaqAPI void UeiDaq::CUeiAnalogScaledReader::ReadSingleScan (f64 * *pBuffer*)

Read only one scan from the input stream

Parameters:

pBuffer destination buffer

2.47.3.2 UeiDaqAPI void UeiDaq::CUeiAnalogScaledReader::ReadMultipleScans (Int32 *numScans*, f64 * *pBuffer*)

Read several scans from the input stream

Parameters:

numScans number of scans to read

pBuffer destination buffer

2.47.3.3 UeiDaqAPI void UeiDaq::CUeiAnalogScaledReader::ReadSingleScanAsync (f64 * *pBuffer*)

Read only one scan asynchronously from the input stream. The destination buffer contains valid data once the listener is called.

Parameters:

pBuffer destination buffer

2.47.3.4 UeiDaqAPI void UeiDaq::CUeiAnalogScaledReader::ReadMultipleScansAsync (Int32 *numScans*, f64 * *pBuffer*)

Read several scans asynchronously from the input stream. The destination buffer contains valid data once the listener is called.

Parameters:

numScans number of scans to read

pBuffer destination buffer

Examples:

AnalogInBufferedAsync.cpp.

2.47.3.5 UeiDaqAPI void UeiDaq::CUeiAnalogScaledReader::AddEventListener (IUeiEventListener * *pListener*)

Subscribe a listener to receive asynchronous events.

Parameters:

pListener pointer to a class that implements **IUeiEventListener** (p. 549) interface

Examples:

AnalogInBufferedAsync.cpp.

The documentation for this class was generated from the following file:

- UeiReader.h

2.48 UeiDaq::CUeiAnalogScaledWriter Class Reference

Analog Scaled Writer class.

`#include <UeiWriter.h>`

Public Member Functions

- UeiDaqAPI **CUeiAnalogScaledWriter** (**CUeiDataStream** *pDataStream)
  Constructor.
- virtual UeiDaqAPI ~**CUeiAnalogScaledWriter** ()
  Destructor.
- UeiDaqAPI void **WriteSingleScan** (**f64** *pBuffer)
  Write a scan.
- UeiDaqAPI void **WriteMultipleScans** (**Int32** numScans, **f64** *pBuffer)
  Write multiple scans.
- UeiDaqAPI void **WriteSingleScanAsync** (**f64** *pBuffer)
  Write a scan asynchronously.
- UeiDaqAPI void **WriteMultipleScansAsync** (**Int32** numScans, **f64** *pBuffer)
  Write multiple scans asynchronously.
- UeiDaqAPI void **AddEventListener** (**IUeiEventListener** *pListener)
  Add an asynchronous listener.

2.48.1 Detailed Description

Class that handles writing scaled data to a stream going to an analog output

Examples:

AnalogOutBuffered.cpp, **AnalogOutBufferedAsync.cpp**, and **AnalogOutSingle.cpp**.

2.48.2 Constructor & Destructor Documentation

2.48.2.1 UeiDaqAPI UeiDaq::CUeiAnalogScaledWriter::CUeiAnalogScaledWriter (CUeiDataStream * *pDataStream*)

Parameters:

pDataStream represents the destination where to write data

2.48.3 Member Function Documentation

2.48.3.1 UeiDaqAPI void UeiDaq::CUeiAnalogScaledWriter::WriteSingleScan (f64 * *pBuffer*)

Write only one scan to the output stream

Parameters:

pBuffer source buffer

2.48.3.2 UeiDaqAPI void UeiDaq::CUeiAnalogScaledWriter::WriteMultipleScans (Int32 *numScans*, f64 * *pBuffer*)

Write several scans to the output stream

Parameters:

numScans number of scans to write

pBuffer source buffer

2.48.3.3 UeiDaqAPI void UeiDaq::CUeiAnalogScaledWriter::WriteSingleScanAsync (f64 * *pBuffer*)

Write only one scan asynchronously to the output stream. The source buffer can be reused once the listener is called.

Parameters:

pBuffer source buffer

2.48.3.4 UeiDaqAPI void UeiDaq::CUeiAnalogScaledWriter::WriteMultipleScansAsync (Int32 *numScans*, f64 * *pBuffer*)

Write several scans asynchronously to the output stream. The source buffer can be reused once the listener is called.

Parameters:

numScans number of scans to write

pBuffer source buffer

Examples:

AnalogOutBufferedAsync.cpp.

2.48.3.5 UeiDaqAPI void UeiDaq::CUeiAnalogScaledWriter::AddEventListener (IUeiEventListener * *pListener*)

Subscribe a listener to receive asynchronous events

Parameters:

pListener pointer to a class that implements **IUeiEventListener** (p. 549) interface

Examples:

AnalogOutBufferedAsync.cpp.

The documentation for this class was generated from the following file:

- UeiWriter.h

2.49 UeiDaq::CUeiAOChannel Class Reference

Manages settings for each voltage Analog output channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Inherited by **UeiDaq::CUeiAOCurrentChannel**, **UeiDaq::CUeiAOProtectedChannel**, **UeiDaq::CUeiAOWaveformChannel**, **UeiDaq::CUeiSimulatedLVDTChannel**, **UeiDaq::CUeiSimulatedRTDChannel**, **UeiDaq::CUeiSimulatedSynchroResolverChannel**, and **UeiDaq::CUeiSimulatedTCChannel**.

Public Member Functions

- UeiDaqAPI **CUeiAOChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiAOChannel** ()
  Destructor.
- UeiDaqAPI **f64 GetMinimum** ()
  Get the minimum value.
- UeiDaqAPI void **SetMinimum** (**f64** minimum)
  Set the minimum value.
- UeiDaqAPI **f64 GetMaximum** ()
  Get the maximum value.
- UeiDaqAPI void **SetMaximum** (**f64** maximum)
  Set the maximum value.
- UeiDaqAPI bool **IsDefaultValueEnabled** (void)
  Get the default value state.
- UeiDaqAPI void **EnableDefaultValue** (bool enableDefaultValue)
  Set the default value state.
- UeiDaqAPI **f64 GetDefaultValue** ()
  Get the default value.
- UeiDaqAPI void **SetDefaultValue** (**f64** defaultVal)
  Set the default value.

2.49.1 Detailed Description

Manages settings for each voltage Analog output channel

2.49.2 Member Function Documentation

2.49.2.1 UeiDaqAPI f64 UeiDaq::CUeiAOChannel::GetMinimum ()

Get the minimum value for the generated signal.

Returns:

the minimum value.

See also:

SetMinimum (p. 87)

2.49.2.2 UeiDaqAPI void UeiDaq::CUeiAOChannel::SetMinimum (f64 *minimum*)

Set the minimum value for the generated signal.

Parameters:

minimum the minimum value.

See also:

GetMinimum (p. 87)

2.49.2.3 UeiDaqAPI f64 UeiDaq::CUeiAOChannel::GetMaximum ()

Get the maximum value for the generated signal.

Returns:

the maximum value.

See also:

SetMaximum (p. 87)

2.49.2.4 UeiDaqAPI void UeiDaq::CUeiAOChannel::SetMaximum (f64 *maximum*)

Set the maximum value for the generated signal.

Parameters:

maximum the maximum value.

See also:

GetMaximum (p. 87)

2.49.2.5 UeiDaqAPI bool UeiDaq::CUeiAOChannel::IsDefaultValueEnabled (void)

Determines whether output will be set to a default value when session stops.

Returns:

The default value state.

See also:

EnableDefaultValue (p. 88)

2.49.2.6 UeiDaqAPI void UeiDaq::CUeiAOChannel::EnableDefaultValue (bool *enableDefaultValue*)

Enables default value. The output will be set to a default value when session stops.

Parameters:

enableDefaultValue true to turn stop value on, false otherwise

See also:

IsDefaultValueEnabled (p. 88)

2.49.2.7 UeiDaqAPI f64 UeiDaq::CUeiAOChannel::GetDefaultValue ()

Get the default value.

Returns:

the default value.

See also:

SetDefaultValue (p. 88)

2.49.2.8 UeiDaqAPI void UeiDaq::CUeiAOChannel::SetDefaultValue (f64 *defaultVal*)

Set the default value.

Parameters:

defaultVal the default value.

See also:

GetDefaultValue (p. 88)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.50 UeiDaq::CUeiAOCurrentChannel Class Reference

Manages settings for each current output channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiAOChannel**.

Public Member Functions

- UeiDaqAPI **CUeiAOCurrentChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiAOCurrentChannel** ()
  Destructor.

2.50.1 Detailed Description

Manages settings for each current output channel

The documentation for this class was generated from the following file:

- UeiChannel.h

2.51 UeiDaq::CUeiAOProtectedChannel Class Reference

Manages settings for each protected analog output channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiAOChannel**.

Inherited by **UeiDaq::CUeiAOProtectedCurrentChannel**.

Public Member Functions

- UeiDaqAPI **CUeiAOProtectedChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiAOProtectedChannel** ()
  Destructor.
- UeiDaqAPI **tUeiAODACMode GetDACMode** ()
  Get the DAC mode.
- UeiDaqAPI void **SetDACMode** (**tUeiAODACMode** mode)
  Set the DAC mode.
- UeiDaqAPI **tUeiAODiagChannel GetADCChannel** (int index)
  Gets diagnostic channels.
- UeiDaqAPI void **SetADCChannel** (int index, **tUeiAODiagChannel** adcChannel)
  Sets diagnostic channels.
- UeiDaqAPI bool **IsCircuitBreakerEnabled** (int index)
  Determines whether the CB is currently protecting the channel.
- UeiDaqAPI void **EnableCircuitBreaker** (int index, bool enable)
  Enable or Disable channel protection.
- UeiDaqAPI double **GetCircuitBreakerLowLimit** (int index)
  Get the minimum circuit breaker limits.
- UeiDaqAPI void **SetCircuitBreakerLowLimit** (int index, double lowLimit)
  Set the minimum circuit breaker limits.
- UeiDaqAPI double **GetCircuitBreakerHighLimit** (int index)
  Get the maximum circuit breaker limits.
- UeiDaqAPI void **SetCircuitBreakerHighLimit** (int index, double highLimit)
  Set the maximum circuit breaker limits.
- UeiDaqAPI double **GetMeasurementRate** (void)
  Get the diagnostic measurement rate.

- UeiDaqAPI void **SetMeasurementRate** (double measurementRate)

  Set the diagnostic measurement rate.

- UeiDaqAPI bool **GetAutoRetry** (void)

  Get the auto retry setting.

- UeiDaqAPI void **SetAutoRetry** (bool autoRetry)

  Set the auto retry setting.

- UeiDaqAPI double **GetAutoRetryRate** (void)

  Get the auto retry rate.

- UeiDaqAPI void **SetAutoRetryRate** (double autoRetryRate)

  Set the auto retry rate.

- UeiDaqAPI **uInt32 GetOverUnderCount** (void)

  Get the over/under count.

- UeiDaqAPI void **SetOverUnderCount** (**uInt32** overUnderCount)

  Set the over/under count.

2.51.1 Detailed Description

Manages settings for each analog output channel protected by a circuit breaker.

2.51.2 Member Function Documentation

2.51.2.1 UeiDaqAPI tUeiAODACMode UeiDaq::CUeiAOProtectedChannel::GetDACMode ()

Each channel uses two DACs that can be enabled independently.

Returns:

The current DAC mode.

2.51.2.2 UeiDaqAPI void UeiDaq::CUeiAOProtectedChannel::SetDACMode (tUeiAODACMode *mode*)

Each channel uses two DACs that can be enabled independently.

Parameters:

mode The new DAC mode.

2.51.2.3 UeiDaqAPI tUeiAODiagChannel UeiDaq::CUeiAOProtectedChannel::GetADCChannel (int *index*)

Each AO channel comes with up to five diagnostic channels. You can select which diagnostic to monitor.

Parameters:

index The 0 based index of the diagnostic channel

Returns:

The diagnostic channel at this index.

2.51.2.4 UeiDaqAPI void UeiDaq::CUeiAOProtectedChannel::SetADCChannel (int *index*, tUeiAODiagChannel *adcChannel*)

Each AO channel comes with up to five diagnostic channels. Add a diagnostic channel to the list of channels

Parameters:

index The 0 based index of the diagnostic channel

adcChannel The new diagnostic channel at specified index

2.51.2.5 UeiDaqAPI bool UeiDaq::CUeiAOProtectedChannel::IsCircuitBreakerEnabled (int *index*)

Return true if circuit breaker for this channel is enabled and false otherwise

Parameters:

index The 0 based index of the circuit breaker

Returns:

Circuit breaker state

2.51.2.6 UeiDaqAPI void UeiDaq::CUeiAOProtectedChannel::EnableCircuitBreaker (int *index*, bool *enable*)

Enable or disable circuit breaker on this channel. when enabled a circuit breaker monitors up a diagnostic channel and opens the circuit if any of the measurements goes out of pre-set limits.

Parameters:

index The 0 based index of the circuit breaker

enable True to turn-on protection, false to turn it off

2.51.2.7 UeiDaqAPI double UeiDaq::CUeiAOProtectedChannel::GetCircuitBreakerLowLimit (int *index*)

Each circuit-breaker can monitor one diagnostic channels. Get the minimum current/volt/temperature allowed on the specified channel. The circuit will open if less than the minimum value is monitored.

Parameters:

index The 0 based index of the circuit breaker

Returns:

The current low limit.

2.51.2.8 UeiDaqAPI void UeiDaq::CUeiAOProtectedChannel::SetCircuitBreakerLowLimit (int *index*, double *lowLimit*)

Each circuit-breaker can monitor one diagnostic channel. Set the minimum current/volt/temperature allowed on the specified channel. The circuit will open if less than the minimum value is monitored.

Parameters:

index The 0 based index of the circuit breaker

lowLimit The new low limit.

2.51.2.9 UeiDaqAPI double UeiDaq::CUeiAOProtectedChannel::GetCircuitBreakerHighLimit (int *index*)

Each circuit-breaker can monitor one diagnostic channel. Get the maximum current/volt/temperature allowed on the specified channel. The circuit will open if more than the maximum value is monitored.

Parameters:

index The 0 based index of the circuit breaker

Returns:

The current high limit.

2.51.2.10 UeiDaqAPI void UeiDaq::CUeiAOProtectedChannel::SetCircuitBreakerHighLimit (int *index*, double *highLimit*)

Each circuit-breaker can monitor one diagnostic channel. Set the maximum current/volt/temperature allowed on the specified channel. The circuit will open if more than the minimum value is monitored.

Parameters:

index The 0 based index of the circuit breaker

highLimit The new high limit.

2.51.2.11 UeiDaqAPI double UeiDaq::CUeiAOProtectedChannel::GetMeasurementRate (void)

Get the rate at which the diagnostic channels are monitored. This rate determines how fast the device react when an under or over limit condition occurs.

Returns:

The current circuit breaker measurement rate.

See also:

SetMeasurementRate (p. 94)

2.51.2.12 UeiDaqAPI void UeiDaq::CUeiAOProtectedChannel::SetMeasurementRate (double *measurementRate*)

Set the rate at which the diagnostic channels are monitored. This rate determines how fast the device react when an under or over limit condition occurs.

Parameters:

measurementRate The new circuit breaker measurement rate.

See also:

GetMeasurementRate (p. 94)

2.51.2.13 UeiDaqAPI bool UeiDaq::CUeiAOProtectedChannel::GetAutoRetry (void)

The auto retry setting specifies whether the device should attempt to close the circuit after it was opened because of an over or under limit condition. If it is set to true the device will try to close the circuit at a rate set by the autoRetryRate setting.

Returns:

true if autoRetry is on, false otherwise.

See also:

SetAutoRetry (p. 94)

2.51.2.14 UeiDaqAPI void UeiDaq::CUeiAOProtectedChannel::SetAutoRetry (bool *autoRetry*)

The auto retry setting specifies whether the device should attempt to close the circuit after it was opened because of an over or under limit condition. If it is set to true the device will try to close the circuit at a rate set by the autoRetryRate setting.

Parameters:

autoRetry true to turn auto-retry on, false to turn it off.

See also:

GetAutoRetry (p. 94)

2.51.2.15 UeiDaqAPI double UeiDaq::CUeiAOProtectedChannel::GetAutoRetryRate (void)

Specifies how often the device will attempt to close a circuit that was opened because of an over or under limit condition.

Returns:

The number of retries per second.

See also:

SetAutoRetryRate (p. 95)

2.51.2.16 UeiDaqAPI void UeiDaq::CUeiAOProtectedChannel::SetAutoRetryRate (double *autoRetryRate*)

Specifies how often the device will attempt to close a circuit that was opened because of an over or under limit condition.

Parameters:

autoRetryRate The new number of retries per second.

See also:

GetAutoRetryRate (p. 95)

2.51.2.17 UeiDaqAPI uInt32 UeiDaq::CUeiAOProtectedChannel::GetOverUnderCount (void)

Specifies number of consecutive over/under limit diagnostic readings that must occur in order to trip breaker.

Returns:

The maximum number of over/under readings.

See also:

SetOverUnderCount (p. 95)

2.51.2.18 UeiDaqAPI void UeiDaq::CUeiAOProtectedChannel::SetOverUnderCount (uInt32 *overUnderCount*)

Specifies number of consecutive over/under limit diagnostic readings that must occur in order to trip breaker.

Parameters:

overUnderCount The new maximum number of over/under readings.

See also:

GetOverUnderCount (p. 95)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.52 UeiDaq::CUeiAOProtectedCurrentChannel Class Reference

Manages settings for each protected analog output channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiAOProtectedChannel**.

Public Member Functions

- UeiDaqAPI **CUeiAOProtectedCurrentChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiAOProtectedCurrentChannel** ()
  Destructor.

2.52.1 Detailed Description

Manages settings for each analog output channel protected by a circuit breaker.

The documentation for this class was generated from the following file:

- UeiChannel.h

2.53 UeiDaq::CUeiAOWaveformChannel Class Reference

Manages settings for each waveform Analog output channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiAOChannel**.

Public Member Functions

- UeiDaqAPI **CUeiAOWaveformChannel** ()

  Constructor.

- virtual UeiDaqAPI ~**CUeiAOWaveformChannel** ()

  Destructor.

- UeiDaqAPI **tUeiAOWaveformClockSource GetMainDACClockSource** ()

  Get the main DAC clock source.

- UeiDaqAPI void **SetMainDACClockSource** (**tUeiAOWaveformClockSource** source)

  Set the main DAC clock source.

- UeiDaqAPI **tUeiAOWaveformOffsetClockSource GetOffsetDACClockSource** ()

  Get the offset DAC clock source.

- UeiDaqAPI void **SetOffsetDACClockSource** (**tUeiAOWaveformOffsetClockSource** source)

  Set the offset DAC clock source.

- UeiDaqAPI **tUeiAOWaveformClockSync GetMainDACClockSync** ()

  Get the main DAC clock synchronization.

- UeiDaqAPI void **SetMainDACClockSync** (**tUeiAOWaveformClockSync** sync)

  Route the main DAC synchronization.

- UeiDaqAPI **tUeiAOWaveformTriggerSource GetMainDACTriggerSource** ()

  Get the main DAC trigger source.

- UeiDaqAPI void **SetMainDACTriggerSource** (**tUeiAOWaveformTriggerSource** source)

  Set the main DAC trigger source.

- UeiDaqAPI **tUeiAOWaveformOffsetTriggerSource GetOffsetDACTriggerSource** ()

  Get the offset DAC trigger source.

- UeiDaqAPI void **SetOffsetDACTriggerSource** (**tUeiAOWaveformOffsetTriggerSource** source)

  Set the offset DAC trigger source.

2.53.1 Detailed Description

Manages settings for each waveform Analog output channel

2.53.2 Member Function Documentation

2.53.2.1 UeiDaqAPI tUeiAOWaveformClockSource UeiDaq::CUeiAOWaveformChannel::GetMainDACClockSource ()

Get the source of the clock used by the main DAC.

Returns:

the main DAC clock source.

See also:

SetMainDACClockSource (p. 99)

2.53.2.2 UeiDaqAPI void UeiDaq::CUeiAOWaveformChannel::SetMainDACClockSource (tUeiAOWaveformClockSource *source*)

Set the source of the clock used by the main DAC.

Parameters:

source the new main DAC clock source.

See also:

GetMainDACClockSource (p. 99)

2.53.2.3 UeiDaqAPI tUeiAOWaveformOffsetClockSource UeiDaq::CUeiAOWaveformChannel::GetOffsetDACClockSource ()

Get the source of the clock used by the offset DAC.

Returns:

the offset DAC clock source.

See also:

SetOffsetDACClockSource (p. 99)

2.53.2.4 UeiDaqAPI void UeiDaq::CUeiAOWaveformChannel::SetOffsetDACClockSource (tUeiAOWaveformOffsetClockSource *source*)

Set the source of the clock used by the offset DAC.

Parameters:

source the new offset DAC clock source.

See also:

GetOffsetDACClockSource (p. 99)

2.53.2.5 UeiDaqAPI tUeiAOWaveformClockSync UeiDaq::CUeiAOWaveformChannel::GetMainDACClockSync ()

Get the output where the clock used by the main DAC is routed.

Returns:

the main DAC clock synchronization.

See also:

SetMainDACClockSync (p. 100)

2.53.2.6 UeiDaqAPI void UeiDaq::CUeiAOWaveformChannel::SetMainDACClockSync (tUeiAOWaveformClockSync *sync*)

Set the output where the clock used by the main DAC will be routed.

Parameters:

sync the new main DAC clock synchronization.

See also:

GetMainDACClockSync (p. 100)

2.53.2.7 UeiDaqAPI tUeiAOWaveformTriggerSource UeiDaq::CUeiAOWaveformChannel::GetMainDACTriggerSource ()

Get the source of the trigger used by the main DAC.

Returns:

the main DAC trigger source.

See also:

SetMainDACTriggerSource (p. 101)

2.53.2.8 UeiDaqAPI void UeiDaq::CUeiAOWaveformChannel::SetMainDACTriggerSource (tUeiAOWaveformTriggerSource *source*)

Set the source of the trigger used by the main DAC.

Parameters:

source the new main DAC trigger source.

See also:

GetMainDACTriggerSource (p. 100)

2.53.2.9 UeiDaqAPI tUeiAOWaveformOffsetTriggerSource UeiDaq::CUeiAOWaveformChannel::GetOffsetDACTriggerSource ()

Get the source of the trigger used by the offset DAC.

Returns:

the offset DAC trigger source.

See also:

SetOffsetDACTriggerSource (p. 101)

2.53.2.10 UeiDaqAPI void UeiDaq::CUeiAOWaveformChannel::SetOffsetDACTriggerSource (tUeiAOWaveformOffsetTriggerSource *source*)

Set the source of the trigger used by the offset DAC.

Parameters:

source the new offset DAC trigger source.

See also:

GetOffsetDACTriggerSource (p. 101)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.54 UeiDaq::CUeiAOWaveformWriter Class Reference

AO Waveform Writer class.

`#include <UeiMessaging.h>`

Public Member Functions

- UeiDaqAPI **CUeiAOWaveformWriter** (**CUeiDataStream** *pDataStream, **Int32** channel=0)

  Constructor.

- virtual UeiDaqAPI ~**CUeiAOWaveformWriter** ()

  Destructor.

- UeiDaqAPI void **WriteWaveform** (**Int32** numWfms, **tUeiAOWaveformParameters** *pBuffer, **Int32** *numWfmsWritten)

  Write waveform to the AO device.

- UeiDaqAPI void **WriteSweep** (**Int32** numSwps, **tUeiAOWaveformSweepParameters** *pBuffer, **Int32** *numSwpsWritten)

  Write sweep command to the AO device.

- UeiDaqAPI void **WriteArbitraryData** (**Int32** numValues, double *pBuffer, **Int32** *numValsWritten)

  Write arbitrary waveform data to the AO device.

2.54.1 Detailed Description

Class that handles writing waveform commands to a stream going through a AO waveform device

2.54.2 Constructor & Destructor Documentation

2.54.2.1 UeiDaqAPI UeiDaq::CUeiAOWaveformWriter::CUeiAOWaveformWriter (CUeiDataStream * *pDataStream*, Int32 *channel* = 0)

Parameters:

pDataStream represents the destination where to write data

channel the AO waveform channel to send command to

2.54.3 Member Function Documentation

2.54.3.1 UeiDaqAPI void UeiDaq::CUeiAOWaveformWriter::WriteWaveform (Int32 *numWfms*, tUeiAOWaveformParameters * *pBuffer*, Int32 * *numWfmsWritten*)

Format and send one or more tUeiAOWaveformParameters(s) to the AO device.

Parameters:

numWfms the number of waveforms to send

pBuffer source buffer

numWfmsWritten the actual number of waveforms sent

2.54.3.2 UeiDaqAPI void UeiDaq::CUeiAOWaveformWriter::WriteSweep (Int32 *numSwps*, tUeiAOWaveformSweepParameters * *pBuffer*, Int32 * *numSwpsWritten*)

Format and send one or more tUeiAOWaveformSweepParameters(s) to the AO device.

Parameters:

numSwps the number of sweep commands to send

pBuffer source buffer

numSwpsWritten the actual number of sweep commands sent

2.54.3.3 UeiDaqAPI void UeiDaq::CUeiAOWaveformWriter::WriteArbitraryData (Int32 *numValues*, double * *pBuffer*, Int32 * *numValsWritten*)

Send arbitrary data to the AO device.

Parameters:

numValues the number of values to send

pBuffer source buffer

numValsWritten the actual number of values sent

The documentation for this class was generated from the following file:

- UeiMessaging.h

2.55 UeiDaq::CUeiARINCInputPort Class Reference

Manages settings for each ARINC-429 input port.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Public Member Functions

- UeiDaqAPI **CUeiARINCInputPort** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiARINCInputPort** ()
  Destructor.
- UeiDaqAPI **tUeiARINCPortSpeed GetSpeed** (void)
  Get the ARINC port speed.
- UeiDaqAPI void **SetSpeed** (**tUeiARINCPortSpeed** bitsPerSecond)
  Set the ARINC port speed.
- UeiDaqAPI **tUeiARINCPortParity GetParity** (void)
  Get the ARINC port parity.
- UeiDaqAPI void **SetParity** (**tUeiARINCPortParity** parity)
  Set the ARINC port parity.
- UeiDaqAPI bool **IsSDIFilterEnabled** (void)
  Get the SDI filter state.
- UeiDaqAPI void **EnableSDIFilter** (bool enableSDIFilter)
  Set the SDI filter state.
- UeiDaqAPI **uInt32 GetSDIFilterMask** (void)
  Get the SDI filter mask.
- UeiDaqAPI void **SetSDIFilterMask** (**uInt32** SDIFilterMask)
  Set the SDI filter mask.
- UeiDaqAPI bool **IsTimestampingEnabled** (void)
  Get the timestamping state.
- UeiDaqAPI void **EnableTimestamping** (bool enableTimestamping)
  Set the timestamping state.
- UeiDaqAPI bool **IsSlowSlewRateEnabled** (void)
  Get the slow slew rate state.
- UeiDaqAPI void **EnableSlowSlewRate** (bool enableSlowSlewRate)

Set the slow slew state.

- UeiDaqAPI void **AddFilterEntry** (**tUeiARINCFilterEntry** entry)
  Add a filter entry.
- UeiDaqAPI **tUeiARINCFilterEntry** * **GetFilterEntry** (int index)
  Retrieve a filter entry.
- UeiDaqAPI void **ClearFilterEntries** (void)
  Clear filter table.
- UeiDaqAPI bool **IsLabelFilterEnabled** (void)
  Get the label filter state.
- UeiDaqAPI void **EnableLabelFilter** (bool enableLabelFilter)
  Set the label filter state.
- UeiDaqAPI **f64 GetTimestampResolution** ()
  Get the timestamp resolution.
- UeiDaqAPI void **SetTimestampResolution** (**f64** resolution)
  Set the timestamp resolution.

2.55.1 Detailed Description

Manages settings for each ARINC-429 input port

2.55.2 Member Function Documentation

2.55.2.1 UeiDaqAPI tUeiARINCPortSpeed UeiDaq::CUeiARINCInputPort::GetSpeed (void)

Get the number of bits transmitted per second.

Returns:

The ARINC port speed.

See also:

SetSpeed (p. 105)

2.55.2.2 UeiDaqAPI void UeiDaq::CUeiARINCInputPort::SetSpeed (tUeiARINCPortSpeed *bitsPerSecond*)

Set the number of bits transmitted per second.

Parameters:

bitsPerSecond The ARINC port speed

See also:

GetSpeed (p. 105)

2.55.2.3 UeiDaqAPI tUeiARINCPortParity UeiDaq::CUeiARINCInputPort::GetParity (void)

Get the parity used to detect transmission errors. The parity bit of each ARINC frame is set to 0 or 1 so that the number of bits set to 1 matches the specified parity.

Returns:

The ARINC port parity.

See also:

SetParity (p. 106)

2.55.2.4 UeiDaqAPI void UeiDaq::CUeiARINCInputPort::SetParity (tUeiARINCPortParity *parity*)

Set the parity used to detect transmission errors. The parity bit of each ARINC frame is set to 0 or 1 so that the number of bits set to 1 matches the specified parity.

Parameters:

parity The ARINC port parity

See also:

GetParity (p. 106)

2.55.2.5 UeiDaqAPI bool UeiDaq::CUeiARINCInputPort::IsSDIFilterEnabled (void)

Determines whether the SDI filter is turned on or off.

Returns:

The SDI filter state.

See also:

EnableSDIFilter (p. 106)

2.55.2.6 UeiDaqAPI void UeiDaq::CUeiARINCInputPort::EnableSDIFilter (bool *enableSDIFilter*)

Specifies whether the SDI filter is turned on or off.

Parameters:

enableSDIFilter true to turn SDI filtering on, false otherwise

See also:

IsSDIFilterEnabled (p. 106)

2.55.2.7 UeiDaqAPI uInt32 UeiDaq::CUeiARINCInputPort::GetSDIFilterMask (void)

Get the SDI filter mask, only bits 0 and 1 are meaningful. ARINC frames whose SDI bits don't match the SDI mask are rejected.

Returns:

The SDI filter mask.

See also:

SetSDIFilterMask (p. 107)

2.55.2.8 UeiDaqAPI void UeiDaq::CUeiARINCInputPort::SetSDIFilterMask (uInt32 *SDIFilterMask*)

Set the SDI filter mask, only bits 0 and 1 are meaningful. ARINC frames whose SDI bits don't match the SDI mask are rejected.

Parameters:

SDIFilterMask The new SDI filter mask

See also:

GetSDIFilterMask (p. 107)

2.55.2.9 UeiDaqAPI bool UeiDaq::CUeiARINCInputPort::IsTimestampingEnabled (void)

Determines whether each received frame is timestamped.

Returns:

The timestamping state.

See also:

EnableTimestamping (p. 107)

2.55.2.10 UeiDaqAPI void UeiDaq::CUeiARINCInputPort::EnableTimestamping (bool *enableTimestamping*)

Specifies whether each received frame is timestamped.

Parameters:

enableTimestamping true to turn timestamping on, false otherwise

See also:

IsTimestampingEnabled (p. 107)

2.55.2.11 UeiDaqAPI bool UeiDaq::CUeiARINCInputPort::IsSlowSlewRateEnabled (void)

Determines whether slow slew rate is enabled.

Returns:

The slow slew rate state.

See also:

EnableSlowSlewRate (p. 108)

2.55.2.12 UeiDaqAPI void UeiDaq::CUeiARINCInputPort::EnableSlowSlewRate (bool *enableSlowSlewRate*)

Enables slow slew rate.

Parameters:

enableSlowSlewRate true to turn slow slew rate on, false otherwise

See also:

IsSlowSlewRateEnabled (p. 108)

2.55.2.13 UeiDaqAPI void UeiDaq::CUeiARINCInputPort::AddFilterEntry (tUeiARINCFilterEntry *entry*)

Add a filter entry to the port's frame filter table. Each incoming ARINC frame that doesn't match any of the filter entries is rejected.

Parameters:

entry A struture that represents the new filter entry

See also:

ClearFilterEntries (p. 109), **GetFilterEntry** (p. 108)

2.55.2.14 UeiDaqAPI tUeiARINCFilterEntry* UeiDaq::CUeiARINCInputPort::GetFilterEntry (int *index*)

Retrieve a filter entry from the port's frame filter table. Returns NULL when retrieving past the end of the table.

Returns:

A struture that represents the retrieved filter entry

See also:

AddFilterEntry (p. 108), **ClearFilterEntries** (p. 109)

2.55.2.15 UeiDaqAPI void UeiDaq::CUeiARINCInputPort::ClearFilterEntries (void)

Empties the port's filter table.

See also:

AddFilterEntry (p. 108), **GetFilterEntry** (p. 108)

2.55.2.16 UeiDaqAPI bool UeiDaq::CUeiARINCInputPort::IsLabelFilterEnabled (void)

Determines whether label filtering is enabled.

Returns:

The label filter state.

See also:

EnableLabelFilter (p. 109)

2.55.2.17 UeiDaqAPI void UeiDaq::CUeiARINCInputPort::EnableLabelFilter (bool *enableLabelFilter*)

Enables label filtering.

Parameters:

enableLabelFilter true to turn label filtering on, false otherwise

See also:

IsLabelFilterEnabled (p. 109)

2.55.2.18 UeiDaqAPI f64 UeiDaq::CUeiARINCInputPort::GetTimestampResolution ()

Get the timestamp resolution in seconds.

Returns:

the timestamp resolution.

See also:

SetTimestampResolution (p. 109)

2.55.2.19 UeiDaqAPI void UeiDaq::CUeiARINCInputPort::SetTimestampResolution (f64 *resolution*)

Set the timestamp resolution in seconds.

Parameters:

resolution the new resolution.

See also:

GetTimestampResolution (p. 109)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.56 UeiDaq::CUeiARINCOutputPort Class Reference

Manages settings for each ARINC-429 output port.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Public Member Functions

- UeiDaqAPI **CUeiARINCOutputPort** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiARINCOutputPort** ()
  Destructor.
- UeiDaqAPI **tUeiARINCPortSpeed GetSpeed** (void)
  Get the ARINC port speed.
- UeiDaqAPI void **SetSpeed** (**tUeiARINCPortSpeed** bitsPerSecond)
  Set the ARINC port speed.
- UeiDaqAPI **tUeiARINCPortParity GetParity** (void)
  Get the ARINC port parity.
- UeiDaqAPI void **SetParity** (**tUeiARINCPortParity** parity)
  Set the ARINC port parity.
- UeiDaqAPI bool **IsLoopbackEnabled** (void)
  Get the loopback state.
- UeiDaqAPI void **EnableLoopback** (bool enableLoopback)
  Set the loopback state.
- UeiDaqAPI bool **IsSlowSlewRateEnabled** (void)
  Get the slow slew rate state.
- UeiDaqAPI void **EnableSlowSlewRate** (bool enableSlowSlewRate)
  Set the slow slew state.
- UeiDaqAPI void **AddSchedulerEntry** (**tUeiARINCSchedulerEntry** entry)
  Add a scheduler entry.
- UeiDaqAPI **tUeiARINCSchedulerEntry** * **GetSchedulerEntry** (int index)
  Retrieve a scheduler entry.
- UeiDaqAPI void **ClearSchedulerEntries** (void)
  Clear scheduler table.
- UeiDaqAPI bool **IsSchedulerEnabled** (void)

Get the scheduler state.

- UeiDaqAPI void **EnableScheduler** (bool enableScheduler)
  Set the scheduler state.
- UeiDaqAPI **tUeiARINCSchedulerType GetSchedulerType** (void)
  Get the scheduler type.
- UeiDaqAPI void **SetSchedulerType** (**tUeiARINCSchedulerType** type)
  Set the scheduler type.
- UeiDaqAPI double **GetSchedulerRate** (void)
  Get the scheduler rate.
- UeiDaqAPI void **SetSchedulerRate** (double rate)
  Set the scheduler rate.
- UeiDaqAPI double **GetFIFORate** (void)
  Get the TX FIFO rate.
- UeiDaqAPI void **SetFIFORate** (double rate)
  Set the TX FIFO rate.
- UeiDaqAPI bool **IsDelayEnabled** (void)
  Get the transmit delay state.
- UeiDaqAPI void **EnableDelay** (bool enableDelay)
  Set the transmit delay state.
- UeiDaqAPI void **AddMinorFrameEntry** (tUeiARINCMinorFrameEntry entry)
  Add a minor frame entry.
- UeiDaqAPI tUeiARINCMinorFrameEntry * **GetMinorFrameEntry** (int index)
  Retrieve a minor frame entry.
- UeiDaqAPI void **ClearMinorFrameEntries** ()
  Clear major frame.

2.56.1 Detailed Description

Manages settings for each ARINC-429 output port

2.56.2 Member Function Documentation

2.56.2.1 UeiDaqAPI tUeiARINCPortSpeed UeiDaq::CUeiARINCOutputPort::GetSpeed (void)

Get the number of bits transmitted per second.

Returns:

The ARINC port speed.

See also:

SetSpeed (p. 113)

2.56.2.2 UeiDaqAPI void UeiDaq::CUeiARINCOutputPort::SetSpeed (tUeiARINCPortSpeed *bitsPerSecond*)

Set the number of bits transmitted per second.

Parameters:

bitsPerSecond The ARINC port speed

See also:

GetSpeed (p. 112)

2.56.2.3 UeiDaqAPI tUeiARINCPortParity UeiDaq::CUeiARINCOutputPort::GetParity (void)

Get the parity used to detect transmission errors. The parity bit of each ARINC frame is set to 0 or 1 so that the number of bits set to 1 matches the specified parity.

Returns:

The ARINC port parity.

See also:

SetParity (p. 113)

2.56.2.4 UeiDaqAPI void UeiDaq::CUeiARINCOutputPort::SetParity (tUeiARINCPortParity *parity*)

Set the parity used to detect transmission errors. The parity bit of each ARINC frame is set to 0 or 1 so that the number of bits set to 1 matches the specified parity.

Parameters:

parity The ARINC port parity

See also:

GetParity (p. 113)

2.56.2.5 UeiDaqAPI bool UeiDaq::CUeiARINCOutputPort::IsLoopbackEnabled (void)

Determines whether loopback is enabled. When enabled you can read back packet sent to this port on a dedicated loopback port.

Returns:

The loopback state.

See also:

EnableLoopback (p. 114)

2.56.2.6 UeiDaqAPI void UeiDaq::CUeiARINCOutputPort::EnableLoopback (bool *enableLoopback*)

Enables loopback. When enabled you can read back packet sent to this port on a dedicated loopback port.

Parameters:

enableLoopback true to enable loopback, false to disable it.

See also:

IsLoopbackEnabled (p. 114)

2.56.2.7 UeiDaqAPI bool UeiDaq::CUeiARINCOutputPort::IsSlowSlewRateEnabled (void)

Determines whether slow slew rate is enabled.

Returns:

The slow slew rate state.

See also:

EnableSlowSlewRate (p. 114)

2.56.2.8 UeiDaqAPI void UeiDaq::CUeiARINCOutputPort::EnableSlowSlewRate (bool *enableSlowSlewRate*)

Enables slow slew rate.

Parameters:

enableSlowSlewRate true to turn slow slew rate on, false otherwise

See also:

IsSlowSlewRateEnabled (p. 114)

2.56.2.9 UeiDaqAPI void UeiDaq::CUeiARINCOutputPort::AddSchedulerEntry (tUeiARINCSchedulerEntry *entry*)

Add a scheduler entry to the port's scheduler table.

Parameters:

entry A struture that represents the new scheduler entry

See also:

ClearSchedulerEntries (p. 115), **GetSchedulerEntry** (p. 115)

2.56.2.10 UeiDaqAPI tUeiARINCSchedulerEntry* UeiDaq::CUeiARINCOutputPort::GetSchedulerEntry (int *index*)

Retrieve a scheduler entry from the port's scheduler table. Returns NULL when retrieving past the end of the table.

Returns:

A struture that represents the retrieved scheduler entry

See also:

AddSchedulerEntry (p. 115), **ClearSchedulerEntries** (p. 115)

2.56.2.11 UeiDaqAPI void UeiDaq::CUeiARINCOutputPort::ClearSchedulerEntries (void)

Empties the port's scheduler table.

See also:

AddSchedulerEntry (p. 115), **GetSchedulerEntry** (p. 115)

2.56.2.12 UeiDaqAPI bool UeiDaq::CUeiARINCOutputPort::IsSchedulerEnabled (void)

Determines whether scheduling is enabled.

Returns:

The scheduler state.

See also:

EnableScheduler (p. 116)

2.56.2.13 UeiDaqAPI void UeiDaq::CUeiARINCOutputPort::EnableScheduler (bool *enableScheduler*)

Enables scheduling.

Parameters:

enableScheduler true to turn scheduling on, false otherwise

See also:

IsSchedulerEnabled (p. 115)

2.56.2.14 UeiDaqAPI tUeiARINCSchedulerType UeiDaq::CUeiARINCOutputPort::GetSchedulerType (void)

Gets scheduler type.

Returns:

The scheduler type.

See also:

SetSchedulerType (p. 116)

2.56.2.15 UeiDaqAPI void UeiDaq::CUeiARINCOutputPort::SetSchedulerType (tUeiARINCSchedulerType *type*)

Sets scheduler type.

Parameters:

type the new scheduler type

See also:

GetSchedulerType (p. 116)

2.56.2.16 UeiDaqAPI double UeiDaq::CUeiARINCOutputPort::GetSchedulerRate (void)

Gets scheduler rate.

Returns:

The scheduler rate.

See also:

SetSchedulerRate (p. 117)

2.56.2.17 UeiDaqAPI void UeiDaq::CUeiARINCOutputPort::SetSchedulerRate (double *rate*)

Sets scheduler rate.

Parameters:

rate the new scheduler rate

See also:

GetSchedulerRate (p. 116)

2.56.2.18 UeiDaqAPI double UeiDaq::CUeiARINCOutputPort::GetFIFORate (void)

Gets TX FIFO rate.

Returns:

The TX FIFO rate.

See also:

SetFIFORate (p. 117)

2.56.2.19 UeiDaqAPI void UeiDaq::CUeiARINCOutputPort::SetFIFORate (double *rate*)

Sets TX FIFO rate.

Parameters:

rate the new TX FIFO rate

See also:

GetFIFORate (p. 117)

2.56.2.20 UeiDaqAPI bool UeiDaq::CUeiARINCOutputPort::IsDelayEnabled (void)

Determines whether delay information can be instered in the transmit stream.

Returns:

The delay state.

See also:

EnableDelay (p. 118)

2.56.2.21 UeiDaqAPI void UeiDaq::CUeiARINCOutputPort::EnableDelay (bool *enableDelay*)

Specifies whether delay information can be inserted in the transmit stream.

Parameters:

enableDelay true to enable delay on, false otherwise

See also:

IsDelayEnabled (p. 117)

2.56.2.22 UeiDaqAPI void UeiDaq::CUeiARINCOutputPort::AddMinorFrameEntry (tUeiARINCMinorFrameEntry *entry*)

Add a minor frame entry to the port's scheduler table.

Parameters:

entry A struture that represents the new minor frame entry

See also:

ClearMinorFrameEntries (p. 118), **GetMinorFrameEntry** (p. 118)

2.56.2.23 UeiDaqAPI tUeiARINCMinorFrameEntry* UeiDaq::CUeiARINCOutputPort::GetMinorFrameEntry (int *index*)

Retrieve a scheduler entry from the port's scheduler table. Returns NULL when retrieving past the end of the table.

Returns:

A struture that represents the retrieved scheduler entry

See also:

AddMinorFrameEntry (p. 118), **ClearMinorFrameEntries** (p. 118)

2.56.2.24 UeiDaqAPI void UeiDaq::CUeiARINCOutputPort::ClearMinorFrameEntries ()

Empties the port's major frame.

See also:

AddMinorFrameEntry (p. 118), **GetMinorFrameEntry** (p. 118)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.57 UeiDaq::CUeiARINCRawReader Class Reference

ARINC-429 Raw Reader class.

`#include <UeiMessaging.h>`

Public Member Functions

- UeiDaqAPI **CUeiARINCRawReader** (**CUeiDataStream** *pDataStream, **Int32** port=0)
  Constructor.
- virtual UeiDaqAPI ~**CUeiARINCRawReader** ()
  Destructor.
- UeiDaqAPI void **Read** (**Int32** numWords, **uInt32** *pBuffer, **Int32** *numWordsRead)
  Read word(s) from the ARINC port.
- UeiDaqAPI void **ReadAsync** (**Int32** numWords, **uInt32** *pBuffer)
  Read word(s) asynchronously from the ARINC port.
- UeiDaqAPI void **AddEventListener** (**IUeiEventListener** *pListener)
  Add an asynchronous listener.

2.57.1 Detailed Description

This class handles reading raw data words of a stream coming from an ARINC-429 interface. It returns raw 32-bit ARINC word(s).

2.57.2 Constructor & Destructor Documentation

2.57.2.1 UeiDaqAPI UeiDaq::CUeiARINCRawReader::CUeiARINCRawReader (CUeiDataStream * *pDataStream*, Int32 *port* = 0)

Parameters:

pDataStream represents the source of data to read

port the ARINC port

2.57.3 Member Function Documentation

2.57.3.1 UeiDaqAPI void UeiDaq::CUeiARINCRawReader::Read (Int32 *numWords*, uInt32 * *pBuffer*, Int32 * *numWordsRead*)

Read raw 32-bit ARINC words from the input stream

Parameters:

numWords the number of words that can be stored in the destination buffer

pBuffer destination buffer

numWordsRead the actual number of frames read from the ARINC port

2.57.3.2 UeiDaqAPI void UeiDaq::CUeiARINCRawReader::ReadAsync (Int32 *numWords*, uInt32 * *pBuffer*)

Read raw 32-bit ARINC words from the input stream

Parameters:

numWords the number of words that can be stored in the destination buffer

pBuffer destination buffer

2.57.3.3 UeiDaqAPI void UeiDaq::CUeiARINCRawReader::AddEventListener (IUeiEventListener * *pListener*)

Subscribe a listener to receive asynchronous events

Parameters:

pListener pointer to a class that implements **IUeiEventListener** (p. 549) interface

The documentation for this class was generated from the following file:

- UeiMessaging.h

2.58 UeiDaq::CUeiARINCRawWriter Class Reference

ARINC-429 Writer class.

`#include <UeiMessaging.h>`

Public Member Functions

- UeiDaqAPI **CUeiARINCRawWriter** (**CUeiDataStream** *pDataStream, **Int32** port=0)
  Constructor.
- virtual UeiDaqAPI ~**CUeiARINCRawWriter** ()
  Destructor.
- UeiDaqAPI void **Write** (**Int32** numWords, **uInt32** *pBuffer, **Int32** *numWordsWritten)
  Write word(s) to the ARINC bus.
- UeiDaqAPI void **WriteScheduler** (**Int32** firstWord, **Int32** numWords, **uInt32** *pBuffer, **Int32** *numWordsWritten)
  Update word in ARINC scheduler table.
- UeiDaqAPI void **WriteAsync** (**Int32** numWords, **uInt32** *pBuffer)
  Write word(s) asynchronously to the ARINC bus.
- UeiDaqAPI void **AddEventListener** (**IUeiEventListener** *pListener)
  Add an asynchronous listener.

2.58.1 Detailed Description

Class that handles writing ARINC-429 words to a stream going through an ARINC-429 interface. It expects raw 32-bit ARINC words.

2.58.2 Constructor & Destructor Documentation

2.58.2.1 UeiDaqAPI UeiDaq::CUeiARINCRawWriter::CUeiARINCRawWriter (CUeiDataStream * *pDataStream*, Int32 *port* = 0)

Parameters:

pDataStream represents the destination where to write data

port the ARINC port to write to

2.58.3 Member Function Documentation

2.58.3.1 UeiDaqAPI void UeiDaq::CUeiARINCRawWriter::Write (Int32 *numWords*, uInt32 * *pBuffer*, Int32 * *numWordsWritten*)

Send one or more raw 32-bit word(s) to the ARINC interface.

Parameters:

numWords the number of words to send

pBuffer source buffer

numWordsWritten the actual number of words sent over the ARINC bus

2.58.3.2 UeiDaqAPI void UeiDaq::CUeiARINCRawWriter::WriteScheduler (Int32 *firstWord*, Int32 *numWords*, uInt32 * *pBuffer*, Int32 * *numWordsWritten*)

Update word in ARINC scheduler table.

Parameters:

firstWord Index of the first word to modify in the scheduler table

numWords the number of words to update in the scheduler

pBuffer source buffer

numWordsWritten the actual number of words sent to the ARINC scheduler

2.58.3.3 UeiDaqAPI void UeiDaq::CUeiARINCRawWriter::WriteAsync (Int32 *numWords*, uInt32 * *pBuffer*)

Send one or more raw 32-bit word(s) to the ARINC interface.

Parameters:

numWords the number of words to send

pBuffer source buffer

2.58.3.4 UeiDaqAPI void UeiDaq::CUeiARINCRawWriter::AddEventListener (IUeiEventListener * *pListener*)

Subscribe a listener to receive asynchronous events

Parameters:

pListener pointer to a class that implements **IUeiEventListener** (p. 549) interface

The documentation for this class was generated from the following file:

- UeiMessaging.h

2.59 UeiDaq::CUeiARINCReader Class Reference

ARINC-429 Reader class.

`#include <UeiMessaging.h>`

Public Member Functions

- UeiDaqAPI **CUeiARINCReader** (**CUeiDataStream** *pDataStream, **Int32** port=0)
  Constructor.
- virtual UeiDaqAPI ~**CUeiARINCReader** ()
  Destructor.
- UeiDaqAPI void **Read** (**Int32** numWords, **tUeiARINCWord** *pBuffer, **Int32** *numWordsRead)
  Read word(s) from the ARINC port.
- UeiDaqAPI void **ReadAsync** (**Int32** numWords, **tUeiARINCWord** *pBuffer)
  Read word(s) asynchronously from the ARINC port.
- UeiDaqAPI void **AddEventListener** (**IUeiEventListener** *pListener)
  Add an asynchronous listener.

2.59.1 Detailed Description

Class that handles reading data words of a stream coming from an ARINC-429 interface. It takes care of converting the raw 32bit ARINC words to ARINC fields such as label, data, SDI and SSM

2.59.2 Constructor & Destructor Documentation

2.59.2.1 UeiDaqAPI UeiDaq::CUeiARINCReader::CUeiARINCReader (CUeiDataStream * *pDataStream*, Int32 *port* = 0)

Parameters:

pDataStream represents the source of data to read

port the ARINC port

2.59.3 Member Function Documentation

2.59.3.1 UeiDaqAPI void UeiDaq::CUeiARINCReader::Read (Int32 *numWords*, tUeiARINCWord * *pBuffer*, Int32 * *numWordsRead*)

Read and parse ARINC words from the input stream

Parameters:

numWords the number of words that can be stored in the destination buffer

pBuffer destination buffer

numWordsRead the actual number of frames read from the ARINC port

2.59.3.2 UeiDaqAPI void UeiDaq::CUeiARINCReader::ReadAsync (Int32 *numWords*, tUeiARINCWord * *pBuffer*)

Read and parse ARINC words from the input stream

Parameters:

numWords the number of words that can be stored in the destination buffer

pBuffer destination buffer

2.59.3.3 UeiDaqAPI void UeiDaq::CUeiARINCReader::AddEventListener (IUeiEventListener * *pListener*)

Subscribe a listener to receive asynchronous events

Parameters:

pListener pointer to a class that implements **IUeiEventListener** (p. 549) interface

The documentation for this class was generated from the following file:

- UeiMessaging.h

2.60 UeiDaq::CUeiARINCWriter Class Reference

ARINC-429 Writer class.

`#include <UeiMessaging.h>`

Public Member Functions

- UeiDaqAPI **CUeiARINCWriter** (**CUeiDataStream** *pDataStream, **Int32** port=0)
  Constructor.
- virtual UeiDaqAPI ~**CUeiARINCWriter** ()
  Destructor.
- UeiDaqAPI void **Write** (**Int32** numWords, **tUeiARINCWord** *pBuffer, **Int32** *numWordsWritten)
  Write formatted word(s) to the ARINC bus.
- UeiDaqAPI void **WriteScheduler** (**Int32** firstWord, **Int32** numWords, **tUeiARINCWord** *pBuffer, **Int32** *numWordsWritten)
  Update word in ARINC scheduler table.
- UeiDaqAPI void **EnableScheduler** (bool enable)
  Enable scheduler.
- UeiDaqAPI void **SetTransmitPage** (bool immediate, **uInt32** writeMask, **uInt32** txMask)
  Set the writing page/transmit page for all channels at once.
- UeiDaqAPI void **WriteAsync** (**Int32** numWords, **tUeiARINCWord** *pBuffer)
  Write formatted word(s) asynchronously to the ARINC bus.
- UeiDaqAPI void **AddEventListener** (**IUeiEventListener** *pListener)
  Add an asynchronous listener.

2.60.1 Detailed Description

Class that handles writing ARINC-429 words to a stream going through an ARINC-429 interface. It takes care of converting ARINC fields such as label, data, SDI and SSM to 32-bit ARINC words.

2.60.2 Constructor & Destructor Documentation

2.60.2.1 UeiDaqAPI UeiDaq::CUeiARINCWriter::CUeiARINCWriter (CUeiDataStream * *pDataStream*, Int32 *port* = 0)

Parameters:

pDataStream represents the destination where to write data

port the ARINC port to write to

2.60.3 Member Function Documentation

2.60.3.1 UeiDaqAPI void UeiDaq::CUeiARINCWriter::Write (Int32 *numWords*, tUeiARINCWord * *pBuffer*, Int32 * *numWordsWritten*)

Format and send one or more word(s) to the ARINC interface.

Parameters:

numWords the number of words to send

pBuffer source buffer

numWordsWritten the actual number of words sent over the ARINC bus

2.60.3.2 UeiDaqAPI void UeiDaq::CUeiARINCWriter::WriteScheduler (Int32 *firstWord*, Int32 *numWords*, tUeiARINCWord * *pBuffer*, Int32 * *numWordsWritten*)

Update word in ARINC scheduler table.

Parameters:

firstWord Index of the first word to modify in the scheduler table

numWords the number of words to update in the scheduler

pBuffer source buffer

numWordsWritten the actual number of words sent to the ARINC scheduler

2.60.3.3 UeiDaqAPI void UeiDaq::CUeiARINCWriter::EnableScheduler (bool *enable*)

Enable or disable scheduler

Parameters:

enable true to enable, false to disable scheduler

2.60.3.4 UeiDaqAPI void UeiDaq::CUeiARINCWriter::SetTransmitPage (bool *immediate*, uInt32 *writeMask*, uInt32 *txMask*)

Major/Minor scheduler features two copies of the scheduler entries. Page 0 and page 1. This call specifies which channel should be written to and which channel should be transmitting. This is designed to configure or update a page while the other is being transmitted. But it is still possible to both write and transmit a channel at the same time.

Parameters:

immediate Specifies when the page configuration will be updated (true to update immediately, false to wait for the next major frame clock)

writeMask Bit mask specifying the writing page for each channel (bit set to 0 for page 0 or 1 for page 1)

txMask Bit mask specifying the transmitting page for each channel (bit set to 0 for page 0 or 1 for page 1)

2.60.3.5 UeiDaqAPI void UeiDaq::CUeiARINCWriter::WriteAsync (Int32 *numWords*, tUeiARINCWord * *pBuffer*)

Format and send one or more word(s) to the ARINC interface.

Parameters:

numWords the number of words to send

pBuffer source buffer

2.60.3.6 UeiDaqAPI void UeiDaq::CUeiARINCWriter::AddEventListener (IUeiEventListener * *pListener*)

Subscribe a listener to receive asynchronous events

Parameters:

pListener pointer to a class that implements **IUeiEventListener** (p. 549) interface

The documentation for this class was generated from the following file:

- UeiMessaging.h

2.61 UeiDaq::CUeiCANPort Class Reference

Manages settings for each CAN port.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Public Member Functions

- UeiDaqAPI **CUeiCANPort** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiCANPort** ()
  Destructor.
- UeiDaqAPI **tUeiCANPortSpeed GetSpeed** ()
  Get the CAN port speed.
- UeiDaqAPI void **SetSpeed** (**tUeiCANPortSpeed** bitsPerSecond)
  Set the CAN port speed.
- UeiDaqAPI **tUeiCANFrameFormat GetFrameFormat** ()
  Get the frame format.
- UeiDaqAPI void **SetFrameFormat** (**tUeiCANFrameFormat** frameFormat)
  Set the frame format.
- UeiDaqAPI **tUeiCANPortMode GetMode** ()
  Get the operation mode.
- UeiDaqAPI void **SetMode** (**tUeiCANPortMode** mode)
  Set the operation mode.
- UeiDaqAPI **uInt32 GetAcceptanceMask** ()
  Get the acceptance mask.
- UeiDaqAPI void **SetAcceptanceMask** (**uInt32** mask)
  Set the acceptance mask.
- UeiDaqAPI **uInt32 GetAcceptanceCode** ()
  Get the acceptance code.
- UeiDaqAPI void **SetAcceptanceCode** (**uInt32** code)
  Set the acceptance code.
- UeiDaqAPI void **AddFilterEntry** (**tUeiCANFilterEntry** entry)
  Add a filter entry.
- UeiDaqAPI **tUeiCANFilterEntry** * **GetFilterEntry** (int index)

Retrieve a filter entry.

- UeiDaqAPI void **ClearFilterEntries** (void)
  Clear filter table.
- UeiDaqAPI bool **IsWarningAndErrorLoggingEnabled** ()
  Determines whether bus errors and warnings are logged.
- UeiDaqAPI void **EnableWarningAndErrorLogging** (bool enable)
  Specifies whether bus errors and warnings are logged.
- UeiDaqAPI **uInt32 GetTransmitErrorCounter** ()
  Get the transmit error counter.
- UeiDaqAPI **uInt32 GetReceiveErrorCounter** ()
  Get the receive error counter.
- UeiDaqAPI **uInt32 GetErrorCodeCaptureRegister** ()
  Get the error code capture register.
- UeiDaqAPI **uInt32 GetArbitrationLostCaptureRegister** ()
  Get the arbitration lost capture register.
- UeiDaqAPI **uInt16 GetBitTimingRegisters** ()
  Get the bit timing registers BTR0 and BTR1.
- UeiDaqAPI void **SetBitTimingRegisters** (**uInt16** btrValues)
  Set the bit timing registers BTR0 and BTR1.
- UeiDaqAPI **f64 GetTimestampResolution** ()
  Get the timestamp resolution.
- UeiDaqAPI void **SetTimestampResolution** (**f64** resolution)
  Set the timestamp resolution.

2.61.1 Detailed Description

Manages settings for each CAN port.

2.61.2 Member Function Documentation

2.61.2.1 UeiDaqAPI tUeiCANPortSpeed UeiDaq::CUeiCANPort::GetSpeed ()

Get the number of data bits transmitted per second.

Returns:

the CAN port speed.

See also:

SetSpeed (p. 130)

2.61.2.2 UeiDaqAPI void UeiDaq::CUeiCANPort::SetSpeed (tUeiCANPortSpeed *bitsPerSecond*)

Set the number of data bits transmitted per second.

Parameters:

bitsPerSecond The port speed

See also:

GetSpeed (p. 129)

2.61.2.3 UeiDaqAPI tUeiCANFrameFormat UeiDaq::CUeiCANPort::GetFrameFormat ()

Get the format of frames transmitted by this port

Returns:

The frame format.

See also:

SetFrameFormat (p. 130)

2.61.2.4 UeiDaqAPI void UeiDaq::CUeiCANPort::SetFrameFormat (tUeiCANFrameFormat *frameFormat*)

Set the format of frames transmitted by this port (CAN frames can be basic or extended).

Parameters:

frameFormat The Frame format

See also:

GetFrameFormat (p. 130)

2.61.2.5 UeiDaqAPI tUeiCANPortMode UeiDaq::CUeiCANPort::GetMode ()

Get the CAN port operation mode. Possible values are normal and passive

Returns:

The operation mode.

See also:

SetMode (p. 131)

2.61.2.6 UeiDaqAPI void UeiDaq::CUeiCANPort::SetMode (tUeiCANPortMode *mode*)

Set the CAN port operation mode. Possible values are normal and passive

Parameters:

mode The operation mode

See also:

GetMode (p. 130)

2.61.2.7 UeiDaqAPI uInt32 UeiDaq::CUeiCANPort::GetAcceptanceMask ()

The acceptance mask programs the hardware filter to reject or accept incoming frames. The mask selects which bits within arbitration ID will be used for filtering.

Returns:

The current acceptance mask.

See also:

SetAcceptanceMask (p. 131)

2.61.2.8 UeiDaqAPI void UeiDaq::CUeiCANPort::SetAcceptanceMask (uInt32 *mask*)

The acceptance mask programs the hardware filter to reject or accept incoming frames. The mask selects which bits within arbitration ID will be used for filtering.

Parameters:

mask The acceptance mask

See also:

GetAcceptanceMask (p. 131)

2.61.2.9 UeiDaqAPI uInt32 UeiDaq::CUeiCANPort::GetAcceptanceCode ()

The acceptance code programs the hardware filter to reject or accept incoming frames. The arbitration ID bits selected by the mask are compared to the code and the frame is rejected if there is any difference. If (mask XOR id) AND code == 0 the frame is accepted

Returns:

The current acceptance code.

See also:

SetAcceptanceCode (p. 132)

2.61.2.10 UeiDaqAPI void UeiDaq::CUeiCANPort::SetAcceptanceCode (uInt32 *code*)

The acceptance code programs the hardware filter to reject or accept incoming frames. The arbitration ID bits selected by the mask are compared to the code and the frame is rejected if there is any difference.

Parameters:

code The acceptance code

See also:

GetAcceptanceCode (p. 131)

2.61.2.11 UeiDaqAPI void UeiDaq::CUeiCANPort::AddFilterEntry (tUeiCANFilterEntry *entry*)

Add a filter entry to the port's filter table. Each incoming CAN frame that doesn't match any of the filter ranges is rejected.

Parameters:

entry A struture that represents the new filter entry

See also:

ClearFilterEntries (p. 132), **GetFilterEntry** (p. 132)

2.61.2.12 UeiDaqAPI tUeiCANFilterEntry* UeiDaq::CUeiCANPort::GetFilterEntry (int *index*)

Retrieve a filter entry from the port's frame filter table. Returns NULL when retrieving past the end of the table.

Returns:

A struture that represents the retrieved filter entry

See also:

AddFilterEntry (p. 132), **ClearFilterEntries** (p. 132)

2.61.2.13 UeiDaqAPI void UeiDaq::CUeiCANPort::ClearFilterEntries (void)

Empties the port's filter table.

See also:

AddFilterEntry (p. 132), **GetFilterEntry** (p. 132)

2.61.2.14 UeiDaqAPI bool UeiDaq::CUeiCANPort::IsWarningAndErrorLoggingEnabled ()

When logging is enabled, bus warnings and errors are sent in the data stream. Use the Type field of the CANDataFrame structure to determine whether received frames are data or error frames.

Returns:

true if logging is enabled, false otherwise.

See also:

EnableWarningAndErrorLogging (p. 133)

2.61.2.15 UeiDaqAPI void UeiDaq::CUeiCANPort::EnableWarningAndErrorLogging (bool *enable*)

When logging is enabled, bus warnings and errors are sent in the data stream. Use the Type field of the CANDataFrame structure to determine whether received frames are data or error frames.

Parameters:

enable true to enable error logging, false to disable it

See also:

IsWarningAndErrorLoggingEnabled (p. 133)

2.61.2.16 UeiDaqAPI uInt32 UeiDaq::CUeiCANPort::GetTransmitErrorCounter ()

Get the number of transmit errors detected on the bus since the session started.

Returns:

The transmit error counter.

See also:

GetReceiveErrorCounter (p. 133)

2.61.2.17 UeiDaqAPI uInt32 UeiDaq::CUeiCANPort::GetReceiveErrorCounter ()

Get the number of receive errors detected on the bus since the session started.

Returns:

The receive error counter.

See also:

GetTransmitErrorCounter (p. 133)

2.61.2.18 UeiDaqAPI uInt32 UeiDaq::CUeiCANPort::GetErrorCodeCaptureRegister ()

Get the current values of the Error Code Capture register from the NXP SJA1000 CAN controller chip. The Error Code Capture register provides information on bus errors that occur according to the CAN standard. A bus error increments either the Transmit Error Counter or the Receive Error Counter. When communication starts on the interface, the first bus error is captured into the Error Code Capture register, and retained until you get this value. Then the Error Code Capture register is again enabled to capture information for the next bus error.

Returns:

The error code capture register.

2.61.2.19 UeiDaqAPI uInt32 UeiDaq::CUeiCANPort::GetArbitrationLostCaptureRegister ()

Get the current values of the Arbitration Lost Capture register from the NXP SJA1000 CAN controller chip.

The Arbitration Lost Capture register provides information on a loss of arbitration during transmits. Loss of arbitration is not considered an error. When communication starts on the interface, the first arbitration loss is captured into the Arbitration Lost Capture register, and retained until you get this value. Then, the Arbitration Lost Capture register is again enabled to capture information for the next arbitration loss.

Returns:

The arbitration lost capture register.

2.61.2.20 UeiDaqAPI uInt16 UeiDaq::CUeiCANPort::GetBitTimingRegisters ()

Get the current values of the bit timing registers from the NXP SJA1000 CAN controller chip. BTR0 and BTR1 set the baud rate for the CAN port. For information on CAN bit timing registers, refer to the datasheet of the NXP SJA1000 CAN controller, available for download on www.nxp.com.

Returns:

The bit timing regiters, BTR0 is LSB and BTR1 is MSB.

2.61.2.21 UeiDaqAPI void UeiDaq::CUeiCANPort::SetBitTimingRegisters (uInt16 *btrValues*)

Set the values of the bit timing registers from the NXP SJA1000 CAN controller chip. BTR0 and BTR1 set the baud rate for the CAN port. For information on CAN bit timing registers, refer to the datasheet of the NXP SJA1000 CAN controller, available for download on www.nxp.com.

Parameters:

btrValues The new bit timing regiters, BTR0 is LSB and BTR1 is MSB.

2.61.2.22 UeiDaqAPI f64 UeiDaq::CUeiCANPort::GetTimestampResolution ()

Get the timestamp resolution in seconds.

Returns:

the timestamp resolution.

See also:

SetTimestampResolution (p. 135)

2.61.2.23 UeiDaqAPI void UeiDaq::CUeiCANPort::SetTimestampResolution (f64 *resolution*)

Set the timestamp resolution in seconds.

Parameters:

resolution the new resolution.

See also:

GetTimestampResolution (p. 135)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.62 UeiDaq::CUeiCANReader Class Reference

CAN bus Reader class.

`#include <UeiMessaging.h>`

Public Member Functions

- UeiDaqAPI **CUeiCANReader** (**CUeiDataStream** *pDataStream, **Int32** port=0)
  Constructor.
- virtual UeiDaqAPI ~**CUeiCANReader** ()
  Destructor.
- UeiDaqAPI void **Read** (**Int32** numFrames, **tUeiCANFrame** *pBuffer, **Int32** *numFrames-Read)
  Read frame(s) from the CAN port.
- UeiDaqAPI void **ReadAsync** (**Int32** numFrames, **tUeiCANFrame** *pBuffer)
  Read frame(s) asynchronously from the CAN port.
- UeiDaqAPI void **AddEventListener** (**IUeiEventListener** *pListener)
  Add an asynchronous listener.

2.62.1 Detailed Description

Class that handles reading data frames of a stream coming from a CAN bus interface

2.62.2 Constructor & Destructor Documentation

2.62.2.1 UeiDaqAPI UeiDaq::CUeiCANReader::CUeiCANReader (CUeiDataStream * *pDataStream*, Int32 *port* = 0)

Parameters:

pDataStream represents the source of data to read

port the CAN port

2.62.3 Member Function Documentation

2.62.3.1 UeiDaqAPI void UeiDaq::CUeiCANReader::Read (Int32 *numFrames*, tUeiCANFrame * *pBuffer*, Int32 * *numFramesRead*)

Read CAN frames from the input stream

Parameters:

numFrames the number of frames that can be stored in the destination buffer

pBuffer destination buffer

numFramesRead the actual number of frames read from the CAN bus

2.62.3.2 UeiDaqAPI void UeiDaq::CUeiCANReader::ReadAsync (Int32 *numFrames*, tUeiCANFrame * *pBuffer*)

Read CAN frames from the input stream

Parameters:

numFrames the number of frames that can be stored in the destination buffer

pBuffer destination buffer

2.62.3.3 UeiDaqAPI void UeiDaq::CUeiCANReader::AddEventListener (IUeiEventListener * *pListener*)

Subscribe a listener to receive asynchronous events

Parameters:

pListener pointer to a class that implements **IUeiEventListener** (p. 549) interface

The documentation for this class was generated from the following file:

- UeiMessaging.h

2.63 UeiDaq::CUeiCANWriter Class Reference

CAN bus Writer class.

`#include <UeiMessaging.h>`

Public Member Functions

- UeiDaqAPI **CUeiCANWriter** (**CUeiDataStream** *pDataStream, **Int32** port=0)
  Constructor.
- virtual UeiDaqAPI ~**CUeiCANWriter** ()
  Destructor.
- UeiDaqAPI void **Write** (**Int32** numFrames, **tUeiCANFrame** *pBuffer, **Int32** *numFrames-Written)
  Write frame(s) to the CAN bus.
- UeiDaqAPI void **WriteAsync** (**Int32** numFrames, **tUeiCANFrame** *pBuffer)
  Write frame(s) asynchronously to the CAN bus.
- UeiDaqAPI void **AddEventListener** (**IUeiEventListener** *pListener)
  Add an asynchronous listener.

2.63.1 Detailed Description

Class that handles writing data frames to a stream going through a CAN bus

2.63.2 Constructor & Destructor Documentation

2.63.2.1 UeiDaqAPI UeiDaq::CUeiCANWriter::CUeiCANWriter (CUeiDataStream * *pDataStream*, Int32 *port* = 0)

Parameters:

pDataStream represents the destination where to write data
port the CAN port to write to

2.63.3 Member Function Documentation

2.63.3.1 UeiDaqAPI void UeiDaq::CUeiCANWriter::Write (Int32 *numFrames*, tUeiCANFrame * *pBuffer*, Int32 * *numFramesWritten*)

Send one or more frames over the CAN bus.

Parameters:

numFrames the number of frames to send
pBuffer source buffer
numFramesWritten the actual number of frames sent over the CAN bus

2.63.3.2 UeiDaqAPI void UeiDaq::CUeiCANWriter::WriteAsync (Int32 *numFrames*, tUeiCANFrame * *pBuffer*)

Send one or more frames over the CAN bus.

Parameters:

numFrames the number of frames to send

pBuffer source buffer

2.63.3.3 UeiDaqAPI void UeiDaq::CUeiCANWriter::AddEventListener (IUeiEventListener * *pListener*)

Subscribe a listener to receive asynchronous events

Parameters:

pListener pointer to a class that implements **IUeiEventListener** (p. 549) interface

The documentation for this class was generated from the following file:

- UeiMessaging.h

2.64 UeiDaq::CUeiChannel Class Reference

Channel base class.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiObject**.

Inherited by **UeiDaq::CUeiAIChannel**, **UeiDaq::CUeiAOChannel**, **UeiDaq::CUeiARINCInputPort**, **UeiDaq::CUeiARINCOutputPort**, **UeiDaq::CUeiCANPort**, **UeiDaq::CUeiCIChannel**, **UeiDaq::CUeiCOChannel**, **UeiDaq::CUeiCSDBPort**, **UeiDaq::CUeiDiagnosticChannel**, **UeiDaq::CUeiDIChannel**, **UeiDaq::CUeiDOChannel**, **UeiDaq::CUeiHDLCPort**, **UeiDaq::CUeiI2CMasterPort**, **UeiDaq::CUeiI2CSlavePort**, **UeiDaq::CUeiIRIGDOTTLChannel**, **UeiDaq::CUeiIRIGInputChannel**, **UeiDaq::CUeiIRIGOutputChannel**, **UeiDaq::CUeiIRIGTimeKeeperChannel**, **UeiDaq::CUeiMIL1553Port**, **UeiDaq::CUeiMuxPort**, **UeiDaq::CUeiSerialPort**, **UeiDaq::CUeiSSIMasterPort**, **UeiDaq::CUeiSSISlavePort**, **UeiDaq::CUeiSync1PPSPort**, **UeiDaq::CUeiTimestampChannel**, and **UeiDaq::CUeiVRChannel**.

Public Member Functions

- UeiDaqAPI **CUeiChannel** ()

  Constructor.

- virtual UeiDaqAPI ~**CUeiChannel** ()

  Destructor.

- UeiDaqAPI std::string **GetResourceName** ()

  Get resource name.

- UeiDaqAPI void **SetResourceName** (std::string resName)

  Set resource name.

- UeiDaqAPI std::string **GetAliasName** ()

  Get alias name.

- UeiDaqAPI void **SetAliasName** (std::string name)

  Set alias name.

- UeiDaqAPI **Int32 GetIndex** ()

  Get the index.

- UeiDaqAPI void **SetIndex** (**Int32** index)

  Set the index.

2.64.1 Detailed Description

Parent class for AI, AO, DI, DO, CI, CO, Serial, CAN, 1555, ARINC-429 channels etc...

2.64.2 Member Function Documentation

2.64.2.1 UeiDaqAPI std::string UeiDaq::CUeiChannel::GetResourceName ()

Get the channel URL. Ex: pwrdaq://Dev1/ai0..

Returns:

A string that contains the resource name.

See also:

SetResourceName (p. 141)

2.64.2.2 UeiDaqAPI void UeiDaq::CUeiChannel::SetResourceName (std::string *resName*)

Set the channel URL. Ex: pwrdaq://Dev1/ai0..

Parameters:

resName A string that contains the resource name.

See also:

GetResourceName (p. 141)

2.64.2.3 UeiDaqAPI std::string UeiDaq::CUeiChannel::GetAliasName ()

Get the alias name of the channel

Returns:

A string that contains the alias name.

See also:

SetAliasName (p. 141)

2.64.2.4 UeiDaqAPI void UeiDaq::CUeiChannel::SetAliasName (std::string *name*)

Set the alias name of the channel

Parameters:

name A string that contains the alias name.

See also:

GetAliasName (p. 141)

2.64.2.5 UeiDaqAPI Int32 UeiDaq::CUeiChannel::GetIndex ()

Get the index of the channel in the session's channel list

Returns:

the channel index.

See also:

SetIndex (p. 142)

2.64.2.6 UeiDaqAPI void UeiDaq::CUeiChannel::SetIndex (Int32 *index*)

Set the index of the channel in the session's channel list

Parameters:

index the channel index.

See also:

GetIndex (p. 142)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.65 UeiDaq::CUeiCIChannel Class Reference

Manages settings for each counter input channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Inherited by **UeiDaq::CUeiQuadratureEncoderChannel**.

Public Member Functions

- UeiDaqAPI **CUeiCIChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiCIChannel** ()
  Destructor.
- UeiDaqAPI **tUeiCounterSource GetCounterSource** ()
  Get the counter source.
- UeiDaqAPI void **SetCounterSource** (**tUeiCounterSource** source)
  Set the counter source.
- UeiDaqAPI **tUeiCounterMode GetCounterMode** ()
  Get the counter mode.
- UeiDaqAPI void **SetCounterMode** (**tUeiCounterMode** mode)
  Set the counter mode.
- UeiDaqAPI **tUeiCounterGate GetCounterGate** ()
  Get the counter gate.
- UeiDaqAPI void **SetCounterGate** (**tUeiCounterGate** gate)
  Set the counter gate.
- UeiDaqAPI **Int32 GetInverted** ()
  Get the inverted setting.
- UeiDaqAPI void **SetInverted** (**Int32** inverted)
  Set the inverted setting.
- UeiDaqAPI **Int32 GetTimebaseDivider** ()
  Get the timebase divider.
- UeiDaqAPI void **SetTimebaseDivider** (**Int32** divider)
  Set the timebase divider.
- UeiDaqAPI double **GetMinimumSourcePulseWidth** ()
  Get the minimum pulse width at the counter source.

- UeiDaqAPI void **SetMinimumSourcePulseWidth** (double minWidth)

  Set the minimum pulse width at the counter source.

- UeiDaqAPI double **GetMinimumGatePulseWidth** ()

  Get the minimum pulse width at the counter gate.

- UeiDaqAPI void **SetMinimumGatePulseWidth** (double minWidth)

  Set the minimum pulse width at the counter gate.

- UeiDaqAPI **tUeiCounterGateMode GetGateMode** ()

  Get the gate mode.

- UeiDaqAPI void **SetGateMode** (**tUeiCounterGateMode** gateMode)

  Set the gate mode.

- UeiDaqAPI **Int32 GetPeriodCount** ()

  Get the period count.

- UeiDaqAPI void **SetPeriodCount** (**Int32** periodCount)

  Set the period count.

- UeiDaqAPI **tUeiCounterCaptureTimebase GetCaptureTimebase** ()

  Get the capture timebase.

- UeiDaqAPI void **SetCaptureTimebase** (**tUeiCounterCaptureTimebase** captureTimebase)

  Set the capture timebase.

- UeiDaqAPI std::string **GetSourcePin** ()

  Get the counter source pin.

- UeiDaqAPI void **SetSourcePin** (std::string pin)

  Set the counter source pin.

- UeiDaqAPI std::string **GetGatePin** ()

  Get the counter gate pin.

- UeiDaqAPI void **SetGatePin** (std::string pin)

  Set the counter gate pin.

- UeiDaqAPI std::string **GetOutputPins** ()

  Get the counter output pin(s).

- UeiDaqAPI void **SetOutputPins** (std::string pins)

  Set the counter output pin.

2.65.1 Detailed Description

Manages settings for each counter input channel

Examples:

MeasureFrequency.cpp.

2.65.2 Member Function Documentation

2.65.2.1 UeiDaqAPI tUeiCounterSource UeiDaq::CUeiCIChannel::GetCounterSource ()

Get the source of the signal counted or used as a timebase

Returns:

the source.

See also:

SetCounterSource (p. 145)

2.65.2.2 UeiDaqAPI void UeiDaq::CUeiCIChannel::SetCounterSource (tUeiCounterSource *source*)

Set the source of the signal counted or used as a timebase

Parameters:

source the source.

See also:

GetCounterSource (p. 145)

2.65.2.3 UeiDaqAPI tUeiCounterMode UeiDaq::CUeiCIChannel::GetCounterMode ()

Get the mode that specify whether the session counts events, measures a pulse width or measures a period on the signal configured as the source

Returns:

the mode.

See also:

SetCounterMode (p. 146)

2.65.2.4 UeiDaqAPI void UeiDaq::CUeiCIChannel::SetCounterMode (tUeiCounterMode *mode*)

Set the mode that specify whether the session counts events, measures a pulse width or measures a period on the signal configured as the source

Parameters:

mode the mode.

See also:

GetCounterMode (p. 145)

2.65.2.5 UeiDaqAPI tUeiCounterGate UeiDaq::CUeiCIChannel::GetCounterGate ()

Get the signal that specify whether the counter/timer is on or off

Returns:

the gate.

See also:

SetCounterGate (p. 146)

2.65.2.6 UeiDaqAPI void UeiDaq::CUeiCIChannel::SetCounterGate (tUeiCounterGate *gate*)

Set the signal that specify whether the counter/timer is on or off

Parameters:

gate the gate.

See also:

GetCounterGate (p. 146)

2.65.2.7 UeiDaqAPI Int32 UeiDaq::CUeiCIChannel::GetInverted ()

Checks whether the signal at the source is inverted before performing the counting operation

Returns:

the inverted setting.

See also:

SetInverted (p. 147)

2.65.2.8 UeiDaqAPI void UeiDaq::CUeiCIChannel::SetInverted (Int32 *inverted*)

Specifies whether the signal at the source is inverted before performing the counting operation

Parameters:

inverted the inverted setting.

See also:

GetInverted (p. 146)

2.65.2.9 UeiDaqAPI Int32 UeiDaq::CUeiCIChannel::GetTimebaseDivider ()

Get the divider used to scale the timebase speed.

Returns:

the divider.

See also:

SetTimebaseDivider (p. 147)

2.65.2.10 UeiDaqAPI void UeiDaq::CUeiCIChannel::SetTimebaseDivider (Int32 *divider*)

Set the divider used to scale the timebase speed.

Parameters:

divider the new divider value.

See also:

GetTimebaseDivider (p. 147)

2.65.2.11 UeiDaqAPI double UeiDaq::CUeiCIChannel::GetMinimumSourcePulseWidth ()

Get the minimum pulse width in ms the counter recognizes at its source. Any pulse whose width is smaller will be ignored.

Returns:

the minimum pulse width

See also:

SetMinimumSourcePulseWidth (p. 148)

2.65.2.12 UeiDaqAPI void UeiDaq::CUeiCIChannel::SetMinimumSourcePulseWidth (double *minWidth*)

Set the minimum pulse width in ms the counter recognizes at its source. Any pulse whose width is smaller will be ignored.

Parameters:

minWidth the minimum pulse width

See also:

GetMinimumSourcePulseWidth (p. 147)

2.65.2.13 UeiDaqAPI double UeiDaq::CUeiCIChannel::GetMinimumGatePulseWidth ()

Get the minimum pulse width in ms the counter recognizes at its gate. Any pulse whose width is smaller will be ignored.

Returns:

the minimum pulse width

See also:

SetMinimumGatePulseWidth (p. 148)

2.65.2.14 UeiDaqAPI void UeiDaq::CUeiCIChannel::SetMinimumGatePulseWidth (double *minWidth*)

Set the minimum pulse width in ms the counter recognizes at its gate. Any pulse whose width is smaller will be ignored.

Parameters:

minWidth the minimum pulse width

See also:

GetMinimumGatePulseWidth (p. 148)

2.65.2.15 UeiDaqAPI tUeiCounterGateMode UeiDaq::CUeiCIChannel::GetGateMode ()

Get the gate mode which determines whether the counter/timer will run continuously once the gate is asserted

Returns:

the gate mode.

See also:

SetGateMode (p. 149)

2.65.2.16 UeiDaqAPI void UeiDaq::CUeiCIChannel::SetGateMode (tUeiCounterGateMode *gateMode*)

Set the gate mode which determines whether the counter/timer will run continuously once the gate is asserted

Parameters:

gateMode the new gate mode.

See also:

GetGateMode (p. 148)

2.65.2.17 UeiDaqAPI Int32 UeiDaq::CUeiCIChannel::GetPeriodCount ()

Get the number of periods used to average a period measurement

Returns:

the period count.

See also:

SetPeriodCount (p. 149)

2.65.2.18 UeiDaqAPI void UeiDaq::CUeiCIChannel::SetPeriodCount (Int32 *periodCount*)

Set the number of period(s) to measure. An average measurement is returned.

Parameters:

periodCount the new period count.

See also:

GetPeriodCount (p. 149)

2.65.2.19 UeiDaqAPI tUeiCounterCaptureTimebase UeiDaq::CUeiCIChannel::GetCaptureTimebase ()

Get the capture timebase which determines the precision of period, frequency and pulse width measurements

Returns:

the the capture timebase.

See also:

SetCaptureTimebase (p. 150)

2.65.2.20 UeiDaqAPI void UeiDaq::CUeiCIChannel::SetCaptureTimebase (tUeiCounterCaptureTimebase *captureTimebase*)

Set the capture timebase which determines the precision of period, frequency and pulse width measurements

Parameters:

captureTimebase the new capture timebase.

See also:

tUeiCounterCaptureTimebase (p. 590)

2.65.2.21 UeiDaqAPI std::string UeiDaq::CUeiCIChannel::GetSourcePin ()

Get the pin used as counter source. The pin name is device dependent Consult the device's user manual to learn about available pins

2.65.2.22 UeiDaqAPI void UeiDaq::CUeiCIChannel::SetSourcePin (std::string *pin*)

Set the pin used as counter source. The pin name is device dependent Consult the device's user manual to learn about available pins

2.65.2.23 UeiDaqAPI std::string UeiDaq::CUeiCIChannel::GetGatePin ()

Get the pin used as counter gate. The pin name is device dependent Consult the device's user manual to learn about available pins

2.65.2.24 UeiDaqAPI void UeiDaq::CUeiCIChannel::SetGatePin (std::string *pin*)

Set the pin used as counter gate. The pin name is device dependent Consult the device's user manual to learn about available pins

2.65.2.25 UeiDaqAPI std::string UeiDaq::CUeiCIChannel::GetOutputPins ()

Get the pin(s) used as counter output. Multiple pins can be used as output when specified in a comma separated list The pin name is device dependent Consult the device's user manual to learn about available pins

2.65.2.26 UeiDaqAPI void UeiDaq::CUeiCIChannel::SetOutputPins (std::string *pins*)

set the pin(s) used as counter output. Multiple pins can be used as output when specified in a comma separated list. The pin name is device dependent Consult the device's user manual to learn about available pins

The documentation for this class was generated from the following file:

- UeiChannel.h

2.66 UeiDaq::CUeiCircuitBreaker Class Reference

Circuit Breaker class.

`#include <UeiMessaging.h>`

Public Member Functions

- UeiDaqAPI **CUeiCircuitBreaker** (**CUeiDataStream** *pDataStream, **Int32** port=0)

  Constructor.

- virtual UeiDaqAPI ~**CUeiCircuitBreaker** ()

  Destructor.

- UeiDaqAPI void **Reset** (**uInt32** mask)

  Resets circuit breaker.

- UeiDaqAPI void **WriteOpenShortSimulation** (**tUeiAOShortOpenCircuitParameters** params)

  Enable/disable short or open state on output devices that support it.

- UeiDaqAPI void **ReadStatus** (**uInt32** *current, **uInt32** *sticky)

  Get current circuit breaker status.

- UeiDaqAPI void **ReadAllStatus** (**Int32** numStatus, **uInt32** *pBuffer, **Int32** *numStatusRead)

  Read detailed status about each circuit breaker.

2.66.1 Detailed Description

Class that handles CB commands to a stream associated with a protected device

2.66.2 Constructor & Destructor Documentation

2.66.2.1 UeiDaqAPI UeiDaq::CUeiCircuitBreaker::CUeiCircuitBreaker (CUeiDataStream * *pDataStream*, Int32 *port* = 0)

Parameters:

pDataStream represents the destination where to access data

port the port on which circuit breakers are manipulated

2.66.3 Member Function Documentation

2.66.3.1 UeiDaqAPI void UeiDaq::CUeiCircuitBreaker::Reset (uInt32 *mask*)

Reset a circuit breaker, in case it was tripped.

Parameters:

mask the mask specifying which circuit breaker to reset (1 to reset, 0 to leave alone)

2.66.3.2 UeiDaqAPI void UeiDaq::CUeiCircuitBreaker::WriteOpenShortSimulation (tUeiAOShortOpenCircuitParameters *params*)

Enable/disable open or short state.

Parameters:

params the open/short simulation parameters

2.66.3.3 UeiDaqAPI void UeiDaq::CUeiCircuitBreaker::ReadStatus (uInt32 * *current*, uInt32 * *sticky*)

Get circuit breaker status masks. Each bit in the current status mask correspond to a circuit breaker. 1 if CB is currently tripped, 0 otherwise Each bit in the sticky status mask correspond to a circuit breaker. 1 if CB was tripped at least once since last time status was read, 0 otherwise

Parameters:

current the current circuit breaker status bits (1 if tripped, 0 otherwise)

sticky the sticky circuit breaker status bits (1 if tripped, 0 otherwise)

2.66.3.4 UeiDaqAPI void UeiDaq::CUeiCircuitBreaker::ReadAllStatus (Int32 *numStatus*, uInt32 * *pBuffer*, Int32 * *numStatusRead*)

Read detailed status about each cicuit breaker. The meaning of each status bits depends on the type of the I/O device. Refer to the device's user manual for detailed description of each status bit

Parameters:

numStatus the number of status to read

pBuffer destination buffer for the channel(s) status

numStatusRead the actual number of status retrieved

The documentation for this class was generated from the following file:

- UeiMessaging.h

2.67 UeiDaq::CUeiCOChannel Class Reference

Manages settings for each frequency output channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Public Member Functions

- UeiDaqAPI **CUeiCOChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiCOChannel** ()
  Destructor.
- UeiDaqAPI **tUeiCounterSource GetCounterSource** ()
  Get the counter source.
- UeiDaqAPI void **SetCounterSource** (**tUeiCounterSource** source)
  Set the counter source.
- UeiDaqAPI **tUeiCounterMode GetCounterMode** ()
  Get the counter mode.
- UeiDaqAPI void **SetCounterMode** (**tUeiCounterMode** mode)
  Set the counter mode.
- UeiDaqAPI **tUeiCounterGate GetCounterGate** ()
  Get the counter gate.
- UeiDaqAPI void **SetCounterGate** (**tUeiCounterGate** gate)
  Set the counter gate.
- UeiDaqAPI **Int32 GetInverted** ()
  Get the inverted setting.
- UeiDaqAPI void **SetInverted** (**Int32** inverted)
  Set the inverted setting.
- UeiDaqAPI **Int32 GetTick1** ()
  Get the number of ticks.
- UeiDaqAPI void **SetTick1** (**Int32** tick1)
  Set the number of ticks.
- UeiDaqAPI **Int32 GetTick2** ()
  Get the number of ticks.
- UeiDaqAPI void **SetTick2** (**Int32** tick2)

Set the number of ticks.

- UeiDaqAPI **Int32 GetTimebaseDivider** ()
  Get the timebase divider.
- UeiDaqAPI void **SetTimebaseDivider** (**Int32** divider)
  Set the timebase divider.
- UeiDaqAPI double **GetMinimumSourcePulseWidth** ()
  Get the minimum pulse width at the counter source.
- UeiDaqAPI void **SetMinimumSourcePulseWidth** (double minWidth)
  Set the minimum pulse width at the counter source.
- UeiDaqAPI double **GetMinimumGatePulseWidth** ()
  Get the minimum pulse width at the counter gate.
- UeiDaqAPI void **SetMinimumGatePulseWidth** (double minWidth)
  Set the minimum pulse width at the counter gate.
- UeiDaqAPI **tUeiCounterGateMode GetGateMode** ()
  Get the gate mode.
- UeiDaqAPI void **SetGateMode** (**tUeiCounterGateMode** gateMode)
  Set the gate mode.
- UeiDaqAPI **Int32 GetNumberOfPulses** ()
  Get the pulse count.
- UeiDaqAPI void **SetNumberOfPulses** (**Int32** numberOfPulses)
  Set the pulse count.
- UeiDaqAPI std::string **GetSourcePin** ()
  Get the counter source pin.
- UeiDaqAPI void **SetSourcePin** (std::string pin)
  Set the counter source pin.
- UeiDaqAPI std::string **GetGatePin** ()
  Get the counter gate pin.
- UeiDaqAPI void **SetGatePin** (std::string pin)
  Set the counter gate pin.
- UeiDaqAPI std::string **GetOutputPins** ()
  Get the counter output pin(s).
- UeiDaqAPI void **SetOutputPins** (std::string pins)
  Set the counter output pin.

2.67.1 Detailed Description

Manages settings for each frequency output channel

2.67.2 Member Function Documentation

2.67.2.1 UeiDaqAPI tUeiCounterSource UeiDaq::CUeiCOChannel::GetCounterSource ()

Get the source of the signal used as a timebase

Returns:

the source.

See also:

GetCounterSource (p. 155)

2.67.2.2 UeiDaqAPI void UeiDaq::CUeiCOChannel::SetCounterSource (tUeiCounterSource *source*)

Set the source of the signal used as a timebase

Parameters:

source the source.

See also:

GetCounterSource (p. 155)

2.67.2.3 UeiDaqAPI tUeiCounterMode UeiDaq::CUeiCOChannel::GetCounterMode ()

Get the mode that specify whether the session generates a single pulse or a pulse train on the counter output

Returns:

the mode.

See also:

SetCounterMode (p. 155)

2.67.2.4 UeiDaqAPI void UeiDaq::CUeiCOChannel::SetCounterMode (tUeiCounterMode *mode*)

Set the mode that specify whether the session generates a single pulse or a pulse train on the counter output

Parameters:

mode the mode.

See also:

GetCounterMode (p. 155)

2.67.2.5 UeiDaqAPI tUeiCounterGate UeiDaq::CUeiCOChannel::GetCounterGate ()

Get the signal that specify whether the counter/timer is on or off

Returns:

the gate.

See also:

SetCounterGate (p. 156)

2.67.2.6 UeiDaqAPI void UeiDaq::CUeiCOChannel::SetCounterGate (tUeiCounterGate *gate*)

Set the signal that specify whether the counter/timer is on or off

Parameters:

gate the gate.

See also:

GetCounterGate (p. 156)

2.67.2.7 UeiDaqAPI Int32 UeiDaq::CUeiCOChannel::GetInverted ()

Checks whether the signal at the source is inverted before performing the timing operation

Returns:

the inverted setting.

See also:

SetInverted (p. 156)

2.67.2.8 UeiDaqAPI void UeiDaq::CUeiCOChannel::SetInverted (Int32 *inverted*)

Specifies whether the signal at the source is inverted before performing the timing operation

Parameters:

inverted the inverted setting.

See also:

GetInverted (p. 156)

2.67.2.9 UeiDaqAPI Int32 UeiDaq::CUeiCOChannel::GetTick1 ()

Specifies how long the output stays low.

Returns:

the number of ticks of the signal connected at the source.

See also:

SetTick1 (p. 157)

2.67.2.10 UeiDaqAPI void UeiDaq::CUeiCOChannel::SetTick1 (Int32 *tick1*)

Specifies how long the output stays low.

Parameters:

tick1 the number of ticks of the signal connected at the source.

See also:

GetTick1 (p. 157)

2.67.2.11 UeiDaqAPI Int32 UeiDaq::CUeiCOChannel::GetTick2 ()

Specifies how long the output stays high.

Returns:

the number of ticks of the signal connected at the source.

See also:

SetTick2 (p. 157)

2.67.2.12 UeiDaqAPI void UeiDaq::CUeiCOChannel::SetTick2 (Int32 *tick2*)

Specifies how long the output stays high.

Parameters:

tick2 the number of ticks of the signal connected at the source.

See also:

GetTick2 (p. 157)

2.67.2.13 UeiDaqAPI Int32 UeiDaq::CUeiCOChannel::GetTimebaseDivider ()

Get the divider used to scale the timebase speed.

Returns:

the divider.

See also:

SetTimebaseDivider (p. 158)

2.67.2.14 UeiDaqAPI void UeiDaq::CUeiCOChannel::SetTimebaseDivider (Int32 *divider*)

Set the divider used to scale the timebase speed.

Parameters:

divider the new divider value.

See also:

GetTimebaseDivider (p. 158)

2.67.2.15 UeiDaqAPI double UeiDaq::CUeiCOChannel::GetMinimumSourcePulseWidth ()

Get the minimum pulse width in ms the counter recognizes at its source. Any pulse whose width is smaller will be ignored.

Returns:

the minimum pulse width

See also:

SetMinimumSourcePulseWidth (p. 158)

2.67.2.16 UeiDaqAPI void UeiDaq::CUeiCOChannel::SetMinimumSourcePulseWidth (double *minWidth*)

Set the minimum pulse width in ms the counter recognizes at its source. Any pulse whose width is smaller will be ignored. Set the minimum pulse width to 0 to disable digital filtering.

Parameters:

minWidth the minimum pulse width

See also:

GetMinimumSourcePulseWidth (p. 158)

2.67.2.17 UeiDaqAPI double UeiDaq::CUeiCOChannel::GetMinimumGatePulseWidth ()

Get the minimum pulse width in ms the counter recognizes at its gate. Any pulse whose width is smaller will be ignored.

Returns:

the minimum pulse width

See also:

SetMinimumGatePulseWidth (p. 159)

2.67.2.18 UeiDaqAPI void UeiDaq::CUeiCOChannel::SetMinimumGatePulseWidth (double *minWidth*)

Set the minimum pulse width in ms the counter recognizes at its gate. Any pulse whose width is smaller will be ignored. Set the minimum width to 0 to disable digital filtering.

Parameters:

minWidth the minimum pulse width

See also:

GetMinimumGatePulseWidth (p. 159)

2.67.2.19 UeiDaqAPI tUeiCounterGateMode UeiDaq::CUeiCOChannel::GetGateMode ()

Get the gate mode which determines whether the counter/timer will run continuously once the gate is asserted

Returns:

the gate mode.

See also:

SetGateMode (p. 159)

2.67.2.20 UeiDaqAPI void UeiDaq::CUeiCOChannel::SetGateMode (tUeiCounterGateMode *gateMode*)

Set the gate mode which determines whether the counter/timer will run continuously once the gate is asserted

Parameters:

gateMode the new gate mode.

See also:

GetGateMode (p. 159)

2.67.2.21 UeiDaqAPI Int32 UeiDaq::CUeiCOChannel::GetNumberOfPulses ()

Get the number of pulses to generate (-1 for continuous)

Returns:

the current number of pulses.

See also:

SetNumberOfPulses (p. 160)

2.67.2.22 UeiDaqAPI void UeiDaq::CUeiCOChannel::SetNumberOfPulses (Int32 *numberOfPulses*)

Set the number of pulses to generate (-1 for continuous)

Parameters:

numberOfPulses the new number of pulses.

See also:

GetNumberOfPulses (p. 160)

2.67.2.23 UeiDaqAPI std::string UeiDaq::CUeiCOChannel::GetSourcePin ()

Get the pin used as counter source. The pin name is device dependent Consult the device's user manual to learn about available pins

2.67.2.24 UeiDaqAPI void UeiDaq::CUeiCOChannel::SetSourcePin (std::string *pin*)

Set the pin used as counter source. The pin name is device dependent Consult the device's user manual to learn about available pins

2.67.2.25 UeiDaqAPI std::string UeiDaq::CUeiCOChannel::GetGatePin ()

Get the pin used as counter gate. The pin name is device dependent Consult the device's user manual to learn about available pins

2.67.2.26 UeiDaqAPI void UeiDaq::CUeiCOChannel::SetGatePin (std::string *pin*)

Set the pin used as counter gate. The pin name is device dependent Consult the device's user manual to learn about available pins

2.67.2.27 UeiDaqAPI std::string UeiDaq::CUeiCOChannel::GetOutputPins ()

Get the pin(s) used as counter output. Multiple pins can be used as output when specified in a comma separated list The pin name is device dependent Consult the device's user manual to learn about available pins

2.67.2.28 UeiDaqAPI void UeiDaq::CUeiCOChannel::SetOutputPins (std::string *pins*)

set the pin(s) used as counter output. Multiple pins can be used as output when specified in a comma separated list. The pin name is device dependent Consult the device's user manual to learn about available pins

The documentation for this class was generated from the following file:

- UeiChannel.h

2.68 UeiDaq::CUeiCounterReader Class Reference

Counter Reader class.

`#include <UeiReader.h>`

Public Member Functions

- UeiDaqAPI **CUeiCounterReader** (**CUeiDataStream** *pDataStream)
  Constructor.
- virtual UeiDaqAPI ~**CUeiCounterReader** ()
  Destructor.
- UeiDaqAPI void **ReadSingleScan** (**uInt16** *pBuffer)
  Read a 16 bits wide scan.
- UeiDaqAPI void **ReadSingleScan** (**uInt32** *pBuffer)
  Read a 32 bits wide scan.
- UeiDaqAPI void **ReadMultipleScans** (**Int32** numScans, **uInt16** *pBuffer)
  Read multiple 16 bits wide scans.
- UeiDaqAPI void **ReadMultipleScans** (**Int32** numScans, **uInt32** *pBuffer)
  Read multiple 32 bits wide scans.
- UeiDaqAPI void **ReadSingleScanAsync** (**uInt16** *pBuffer)
  Read a 16 bits wide scan asynchronously.
- UeiDaqAPI void **ReadSingleScanAsync** (**uInt32** *pBuffer)
  Read a 32 bits wide scan asynchronously.
- UeiDaqAPI void **ReadMultipleScansAsync** (**Int32** numScans, **uInt16** *pBuffer)
  Read multiple 16 bits wide scans asynchronously.
- UeiDaqAPI void **ReadMultipleScansAsync** (**Int32** numScans, **uInt32** *pBuffer)
  Read multiple 32 bits wide scans asynchronously.
- UeiDaqAPI void **AddEventListener** (**IUeiEventListener** *pListener)
  Add an asynchronous listener.

2.68.1 Detailed Description

Class that handles reading data of a stream coming from a Counter input

Examples:

BufferedEventCounting.cpp, **EventCount.cpp**, and **MeasureFrequency.cpp**.

2.68.2 Constructor & Destructor Documentation

2.68.2.1 UeiDaqAPI UeiDaq::CUeiCounterReader::CUeiCounterReader (CUeiDataStream * *pDataStream*)

Parameters:

pDataStream represents the source of data to read

2.68.3 Member Function Documentation

2.68.3.1 UeiDaqAPI void UeiDaq::CUeiCounterReader::ReadSingleScan (uInt16 * *pBuffer*)

Read only one scan from the input stream

Parameters:

pBuffer destination buffer

2.68.3.2 UeiDaqAPI void UeiDaq::CUeiCounterReader::ReadSingleScan (uInt32 * *pBuffer*)

Read only one scan from the input stream

Parameters:

pBuffer destination buffer

2.68.3.3 UeiDaqAPI void UeiDaq::CUeiCounterReader::ReadMultipleScans (Int32 *numScans*, uInt16 * *pBuffer*)

Read several scans from the input stream

Parameters:

numScans number of scans to read

pBuffer destination buffer

2.68.3.4 UeiDaqAPI void UeiDaq::CUeiCounterReader::ReadMultipleScans (Int32 *numScans*, uInt32 * *pBuffer*)

Read several scans from the input stream

Parameters:

numScans number of scans to read

pBuffer destination buffer

2.68.3.5 UeiDaqAPI void UeiDaq::CUeiCounterReader::ReadSingleScanAsync (uInt16 * *pBuffer*)

Read only one scan asynchronously from the input stream. The destination buffer contains valid data once the listener is called.

Parameters:

pBuffer destination buffer

2.68.3.6 UeiDaqAPI void UeiDaq::CUeiCounterReader::ReadSingleScanAsync (uInt32 * *pBuffer*)

Read only one scan asynchronously from the input stream. The destination buffer contains valid data once the listener is called.

Parameters:

pBuffer destination buffer

2.68.3.7 UeiDaqAPI void UeiDaq::CUeiCounterReader::ReadMultipleScansAsync (Int32 *numScans*, uInt16 * *pBuffer*)

Read several scans asynchronously from the input stream. The destination buffer contains valid data once the listener is called.

Parameters:

numScans number of scans to read

pBuffer destination buffer

2.68.3.8 UeiDaqAPI void UeiDaq::CUeiCounterReader::ReadMultipleScansAsync (Int32 *numScans*, uInt32 * *pBuffer*)

Read several scans asynchronously from the input stream. The destination buffer contains valid data once the listener is called.

Parameters:

numScans number of scans to read

pBuffer destination buffer

2.68.3.9 UeiDaqAPI void UeiDaq::CUeiCounterReader::AddEventListener (IUeiEventListener * *pListener*)

Subscribe a listener to receive asynchronous events

Parameters:

pListener pointer to a class that implements **IUeiEventListener** (p. 549) interface

The documentation for this class was generated from the following file:

- UeiReader.h

2.69 UeiDaq::CUeiCounterWriter Class Reference

Counter Writer class.

`#include <UeiWriter.h>`

Public Member Functions

- UeiDaqAPI **CUeiCounterWriter** (**CUeiDataStream** *pDataStream)
  Constructor.
- virtual UeiDaqAPI ~**CUeiCounterWriter** ()
  Destructor.
- UeiDaqAPI void **WriteSingleScan** (**uInt16** *pBuffer)
  Write a 16 bits wide scan.
- UeiDaqAPI void **WriteSingleScan** (**uInt32** *pBuffer)
  Write a 32 bits wide scan.
- UeiDaqAPI void **WriteMultipleScans** (**Int32** numScans, **uInt16** *pBuffer)
  Write multiple 16 bits wide scans.
- UeiDaqAPI void **WriteMultipleScans** (**Int32** numScans, **uInt32** *pBuffer)
  Write multiple 32 bits wide scans.
- UeiDaqAPI void **WriteSingleScanAsync** (**uInt16** *pBuffer)
  Write a 16 bits wide scan asynchronously.
- UeiDaqAPI void **WriteSingleScanAsync** (**uInt32** *pBuffer)
  Write a 32 bits wide scan asynchronously.
- UeiDaqAPI void **WriteMultipleScansAsync** (**Int32** numScans, **uInt16** *pBuffer)
  Write multiple 16 bits wide scans asynchronously.
- UeiDaqAPI void **WriteMultipleScansAsync** (**Int32** numScans, **uInt32** *pBuffer)
  Write multiple 32 bits wide scans asynchronously.
- UeiDaqAPI void **AddEventListener** (**IUeiEventListener** *pListener)
  Add an asynchronous listener.

2.69.1 Detailed Description

Class that handles writing data to a stream going to a counter output

Examples:

GeneratePulseTrain.cpp.

2.69.2 Constructor & Destructor Documentation

2.69.2.1 UeiDaqAPI UeiDaq::CUeiCounterWriter::CUeiCounterWriter (CUeiDataStream * *pDataStream*)

Parameters:

pDataStream represents the destination where to write data

2.69.3 Member Function Documentation

2.69.3.1 UeiDaqAPI void UeiDaq::CUeiCounterWriter::WriteSingleScan (uInt16 * *pBuffer*)

Write only one scan to the output stream

Parameters:

pBuffer source buffer

2.69.3.2 UeiDaqAPI void UeiDaq::CUeiCounterWriter::WriteSingleScan (uInt32 * *pBuffer*)

Write only one scan to the output stream

Parameters:

pBuffer source buffer

2.69.3.3 UeiDaqAPI void UeiDaq::CUeiCounterWriter::WriteMultipleScans (Int32 *numScans*, uInt16 * *pBuffer*)

Write several scans to the output stream

Parameters:

numScans number of scans to write

pBuffer source buffer

2.69.3.4 UeiDaqAPI void UeiDaq::CUeiCounterWriter::WriteMultipleScans (Int32 *numScans*, uInt32 * *pBuffer*)

Write several scans to the output stream

Parameters:

numScans number of scans to write

pBuffer source buffer

2.69.3.5 UeiDaqAPI void UeiDaq::CUeiCounterWriter::WriteSingleScanAsync (uInt16 * *pBuffer*)

Write only one scan asynchronously to the output stream. The source buffer can be reused once the listener is called.

Parameters:

pBuffer source buffer

2.69.3.6 UeiDaqAPI void UeiDaq::CUeiCounterWriter::WriteSingleScanAsync (uInt32 * *pBuffer*)

Write only one scan asynchronously to the output stream. The source buffer can be reused once the listener is called.

Parameters:

pBuffer source buffer

2.69.3.7 UeiDaqAPI void UeiDaq::CUeiCounterWriter::WriteMultipleScansAsync (Int32 *numScans*, uInt16 * *pBuffer*)

Write several scans asynchronously to the output stream. The source buffer can be reused once the listener is called.

Parameters:

numScans number of scans to write
pBuffer source buffer

2.69.3.8 UeiDaqAPI void UeiDaq::CUeiCounterWriter::WriteMultipleScansAsync (Int32 *numScans*, uInt32 * *pBuffer*)

Write several scans asynchronously to the output stream. The source buffer can be reused once the listener is called.

Parameters:

numScans number of scans to write
pBuffer source buffer

2.69.3.9 UeiDaqAPI void UeiDaq::CUeiCounterWriter::AddEventListener (IUeiEventListener * *pListener*)

Subscribe a listener to receive asynchronous events

Parameters:

pListener pointer to a class that implements **IUeiEventListener** (p. 549) interface

The documentation for this class was generated from the following file:

- UeiWriter.h

2.70 UeiDaq::CUeiCSDBPort Class Reference

Manage CSDB port settings.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Public Member Functions

- UeiDaqAPI **CUeiCSDBPort** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiCSDBPort** ()
  Destructor.
- UeiDaqAPI int **GetBps** ()
  Get port speed in bits per second.
- UeiDaqAPI void **SetBps** (int bps)
  Set port speed in bits per second.
- UeiDaqAPI int **GetParity** ()
  Get port parity.
- UeiDaqAPI void **SetParity** (int parity)
  Set port parity.
- UeiDaqAPI int **GetBlockSize** ()
  Get message block size.
- UeiDaqAPI void **SetBlockSize** (int blockSize)
  Set message block size.
- UeiDaqAPI int **GetNumMessages** ()
  Get the number of message blocks.
- UeiDaqAPI void **SetNumMessages** (int numMessages)
  Get the number of message blocks.
- UeiDaqAPI int **GetInterByteDelayUs** ()
  Get the inter-byte delay.
- UeiDaqAPI void **SetInterByteDelayUs** (int interByteDelay)
  Set the inter-byte delay.
- UeiDaqAPI int **GetInterBlockDelayUs** ()
  Get the inter-block delay.
- UeiDaqAPI void **SetInterBlockDelayUs** (int interBlockDelay)

Set the inter-block delay.

- UeiDaqAPI int **GetFramePeriodUs** ()
  Get the frame period.
- UeiDaqAPI void **SetFramePeriodUs** (int framePeriod)
  Set the frame period.

2.70.1 Detailed Description

This objects stores CSDB port settings

2.70.2 Member Function Documentation

2.70.2.1 UeiDaqAPI int UeiDaq::CUeiCSDBPort::GetBps ()

Get the current port speed in bits per second (12500 or 50000)

Returns:

the current port speed

2.70.2.2 UeiDaqAPI void UeiDaq::CUeiCSDBPort::SetBps (int *bps*)

Set the port speed in bits per second (12500 or 50000)

Parameters:

bps the new port speed

2.70.2.3 UeiDaqAPI int UeiDaq::CUeiCSDBPort::GetParity ()

Get the current port parity (0=even >0=odd)

Returns:

the current port parity

2.70.2.4 UeiDaqAPI void UeiDaq::CUeiCSDBPort::SetParity (int *parity*)

Set the port parity (0=even >0=odd)

Parameters:

parity the new port parity

2.70.2.5 UeiDaqAPI int UeiDaq::CUeiCSDBPort::GetBlockSize ()

Get the current message block size in bytes (including address and status bytes)

Returns:

the current message block size

2.70.2.6 UeiDaqAPI void UeiDaq::CUeiCSDBPort::SetBlockSize (int *blockSize*)

Set the message block size in bytes (including address and status bytes)

Parameters:

blockSize the new message block size

2.70.2.7 UeiDaqAPI int UeiDaq::CUeiCSDBPort::GetNumMessages ()

Get the current number of message blocks within a frame

Returns:

the current number of message blocks

2.70.2.8 UeiDaqAPI void UeiDaq::CUeiCSDBPort::SetNumMessages (int *numMessages*)

Get the current number of message blocks within a frame

Parameters:

numMessages the new number of message blocks

2.70.2.9 UeiDaqAPI int UeiDaq::CUeiCSDBPort::GetInterByteDelayUs ()

Get the current inter-byte delay in usecs

Returns:

the current inter-byte delay

2.70.2.10 UeiDaqAPI void UeiDaq::CUeiCSDBPort::SetInterByteDelayUs (int *interByteDelay*)

Set the new inter-byte delay in usecs

Parameters:

interByteDelay the new inter-byte delay

2.70.2.11 UeiDaqAPI int UeiDaq::CUeiCSDBPort::GetInterBlockDelayUs ()

Get the current inter-block delay in usecs

Returns:

the current inter-block delay

2.70.2.12 UeiDaqAPI void UeiDaq::CUeiCSDBPort::SetInterBlockDelayUs (int *interBlockDelay*)

Set the new inter-block delay in usecs

Parameters:

interBlockDelay the new inter-block delay

2.70.2.13 UeiDaqAPI int UeiDaq::CUeiCSDBPort::GetFramePeriodUs ()

Get the frame period in usecs

Returns:

the current frame period

2.70.2.14 UeiDaqAPI void UeiDaq::CUeiCSDBPort::SetFramePeriodUs (int *framePeriod*)

Set the frame period in usecs

Parameters:

framePeriod the new frame period

The documentation for this class was generated from the following file:

- UeiChannel.h

2.71 UeiDaq::CUeiCSDBReader Class Reference

CSDB reader class.

`#include <UeiMessaging.h>`

Public Member Functions

- UeiDaqAPI **CUeiCSDBReader** (**CUeiDataStream** *pDataStream, **Int32** channel=0)
  Constructor.
- virtual UeiDaqAPI ~**CUeiCSDBReader** ()
  Destructor.
- UeiDaqAPI void **ReadFrame** (**Int32** numVals, **tUeiCSDBMessage** *pBuffer, **Int32** *numValsRead)
  Read entire frame.
- UeiDaqAPI void **ReadMessageByIndex** (**Int32** index, **tUeiCSDBMessage** *pBuffer)
  Read message block by index.
- UeiDaqAPI void **ReadMessageByAddress** (**uInt8** address, **tUeiCSDBMessage** *pBuffer)
  Read message block by address.

2.71.1 Detailed Description

Class that handles reading CSDB frames from a stream associated with a CSDB input device

2.71.2 Constructor & Destructor Documentation

2.71.2.1 UeiDaqAPI UeiDaq::CUeiCSDBReader::CUeiCSDBReader (CUeiDataStream * *pDataStream*, Int32 *channel* = 0)

Parameters:

pDataStream represents the source where to read data

channel the CSDB channel to read from

2.71.3 Member Function Documentation

2.71.3.1 UeiDaqAPI void UeiDaq::CUeiCSDBReader::ReadFrame (Int32 *numVals*, tUeiCSDBMessage * *pBuffer*, Int32 * *numValsRead*)

Read latest frame.

Parameters:

numVals the number of messages to read

pBuffer destination buffer

numValsRead the actual number of messages read

2.71.3.2 UeiDaqAPI void UeiDaq::CUeiCSDBReader::ReadMessageByIndex (Int32 *index*, tUeiCSDBMessage * *pBuffer*)

Read latest received message block selected by index.

Parameters:

index the index of the message to read

pBuffer destination buffer

2.71.3.3 UeiDaqAPI void UeiDaq::CUeiCSDBReader::ReadMessageByAddress (uInt8 *address*, tUeiCSDBMessage * *pBuffer*)

Read latest received message selected block by address.

Parameters:

address the address of of the message to read

pBuffer destination buffer

The documentation for this class was generated from the following file:

- UeiMessaging.h

2.72 UeiDaq::CUeiCSDBWriter Class Reference

CSDB writer class.

`#include <UeiMessaging.h>`

Public Member Functions

- UeiDaqAPI **CUeiCSDBWriter** (**CUeiDataStream** *pDataStream, **Int32** channel=0)
  Constructor.
- virtual UeiDaqAPI ~**CUeiCSDBWriter** ()
  Destructor.
- UeiDaqAPI void **WriteFrame** (**Int32** numVals, **tUeiCSDBMessage** *pBuffer, **Int32** *numValsWritten)
  Write entire frame.
- UeiDaqAPI void **WriteMessageByIndex** (**Int32** index, **tUeiCSDBMessage** *pBuffer)
  Write message block by index.

2.72.1 Detailed Description

Class that handles writing CSDB frames to a stream associated with a CSDB output device

2.72.2 Constructor & Destructor Documentation

2.72.2.1 UeiDaqAPI UeiDaq::CUeiCSDBWriter::CUeiCSDBWriter (CUeiDataStream * *pDataStream*, Int32 *channel* = 0)

Parameters:

pDataStream represents the source where to read data

channel the CSDB channel to write to

2.72.3 Member Function Documentation

2.72.3.1 UeiDaqAPI void UeiDaq::CUeiCSDBWriter::WriteFrame (Int32 *numVals*, tUeiCSDBMessage * *pBuffer*, Int32 * *numValsWritten*)

Write all message blocks of a frame.

Parameters:

numVals the number of messages to write

pBuffer source buffer

numValsWritten the actual number of messages written

2.72.3.2 UeiDaqAPI void UeiDaq::CUeiCSDBWriter::WriteMessageByIndex (Int32 *index*, tUeiCSDBMessage * *pBuffer*)

Update a message block selected by index.

Parameters:

index the index of the message to write or update

pBuffer source buffer

The documentation for this class was generated from the following file:

- UeiMessaging.h

2.73 UeiDaq::CUeiDataStream Class Reference

Stream class.

`#include <UeiDataStream.h>`

Inherits **UeiDaq::CUeiObject**.

Public Member Functions

- UeiDaqAPI **CUeiDataStream** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiDataStream** ()
  Destructor.
- UeiDaqAPI **Int32 GetNumberOfScans** ()
  Get the number of scans to read at a time.
- UeiDaqAPI void **SetNumberOfScans** (**Int32** numScans)
  Set the number of scans to read at a time.
- UeiDaqAPI **Int32 GetScanSize** ()
  Get the size of a scan.
- UeiDaqAPI void **SetScanSize** (**Int32** scanSize)
  Set the size of a scan.
- UeiDaqAPI **Int32 GetSampleSize** ()
  Get the size of a raw sample.
- UeiDaqAPI **Int32 GetNumberOfFrames** ()
  Get the number of frames.
- UeiDaqAPI void **SetNumberOfFrames** (**Int32** numFrames)
  Set the number of frames.
- UeiDaqAPI **Int32 GetRegenerate** ()
  Get the regenerate setting.
- UeiDaqAPI void **SetRegenerate** (**Int32** regen)
  Set the regenerate setting.
- UeiDaqAPI **Int32 GetBurst** ()
  Get the burst setting.
- UeiDaqAPI void **SetBurst** (**Int32** burst)
  Set the burst setting.
- UeiDaqAPI **Int32 GetOverUnderRun** ()

Get the Overrun/underrun setting.

- UeiDaqAPI void **SetOverUnderRun** (**Int32** overUnderRun)
  Set the Overrun/underrun setting.
- UeiDaqAPI bool **IsBlockingEnabled** ()
  Get the value of the blocking setting.
- UeiDaqAPI void **EnableBlocking** (bool blocking)
  Set the value of the blocking setting.
- UeiDaqAPI **Int32 GetCurrentScan** ()
  Get the current scan position.
- UeiDaqAPI **Int32 GetAvailableScans** ()
  Get the number of available scans.
- UeiDaqAPI **Int32 GetTotalScans** ()
  Get the total number of scans.
- UeiDaqAPI void **SetOffset** (**Int32** offset)
  Change the pointer position.
- UeiDaqAPI **tUeiDataStreamRelativeTo GetRelativeTo** ()
  Get the reference position.
- UeiDaqAPI void **SetRelativeTo** (**tUeiDataStreamRelativeTo** relativeTo)
  Set the reference position.
- UeiDaqAPI bool **IsStreamActive** ()
  Get the stream status.
- UeiDaqAPI void **ReadRawData** (**Int32** numScans, **uInt16** *pBuffer, **IUeiEventListener** *pListener=NULL)
  Read 16 bits wide raw scans from the stream.
- UeiDaqAPI void **ReadRawData** (**Int32** numScans, **uInt32** *pBuffer, **IUeiEventListener** *pListener=NULL)
  Read 32 bits wide raw scans from the stream.
- UeiDaqAPI void **ReadCalibratedRawData** (**Int32** numScans, **uInt16** *pBuffer, **IUeiEventListener** *pListener=NULL)
  Read 16 bits wide calibrated raw scans from the stream.
- UeiDaqAPI void **ReadCalibratedRawData** (**Int32** numScans, **uInt32** *pBuffer, **IUeiEventListener** *pListener=NULL)
  Read 32 bits wide calibrated raw scans from the stream.
- UeiDaqAPI void **ReadScaledData** (**Int32** numScans, **f64** *pBuffer, **IUeiEventListener** *pListener=NULL)

Read scaled scans from the stream.

- UeiDaqAPI void **WriteRawData** (**Int32** numScans, **uInt16** *pBuffer, **IUeiEventListener** *pListener=NULL)

  Write 16 bits wide raw scans to the stream.

- UeiDaqAPI void **WriteRawData** (**Int32** numScans, **uInt32** *pBuffer, **IUeiEventListener** *pListener=NULL)

  Write 32 bits raw scans to the stream.

- UeiDaqAPI void **WriteScaledData** (**Int32** numScans, **f64** *pBuffer, **IUeiEventListener** *pListener=NULL)

  Write scaled scans to the stream.

- UeiDaqAPI void **ReadMessage** (**Int32** port, **Int32** messageType, **Int32** bufferSize, **Int8** *pBuffer, **Int32** *numBytesRead, **IUeiEventListener** *pListener=NULL)

  Read a data message from the stream.

- UeiDaqAPI void **WriteMessage** (**Int32** port, **Int32** messageType, **Int32** bufferSize, **Int8** *pBuffer, **Int32** *numBytesWritten, **IUeiEventListener** *pListener=NULL)

  Write a data message to the stream.

- UeiDaqAPI void **ReadARINCWords** (**Int32** port, **Int32** bufferSize, **tUeiARINCWord** *pBuffer, **Int32** *numWordsRead, **IUeiEventListener** *pListener=NULL)

  Read ARINC words from the stream.

- UeiDaqAPI void **WriteARINCWords** (**Int32** port, **Int32** bufferSize, **tUeiARINCWord** *pBuffer, **Int32** *numWordsWritten, **IUeiEventListener** *pListener=NULL)

  Write ARINC words to the stream.

- UeiDaqAPI void **WriteScheduledARINCWords** (**Int32** port, **Int32** firstWord, **Int32** bufferSize, **tUeiARINCWord** *pBuffer, **Int32** *numWordsWritten, **IUeiEventListener** *pListener=NULL)

  Write scheduled ARINC words.

- UeiDaqAPI void **WriteScheduledARINCRawWords** (**Int32** port, **Int32** firstWord, **Int32** bufferSize, **uInt32** *pBuffer, **Int32** *numWordsWritten, **IUeiEventListener** *pListener=NULL)

  Write scheduled ARINC raw words.

- UeiDaqAPI void **Flush** (**Int32** port, **tUeiFlushAction** action)

  Flush I/O buffers associated with the specifed port.

- UeiDaqAPI **Int32 GetAvailableInputMessages** (**Int32** port)

  Get the number of available input messages.

- UeiDaqAPI **Int32 GetAvailableOutputSlots** (**Int32** port)

  Get the number of available output slots.

- UeiDaqAPI **Int32 GetTotalMessages** (**Int32** port)

  Get the total number of messages received.

2.73.1 Detailed Description

Represents the data streaming in or out of the device

2.73.2 Member Function Documentation

2.73.2.1 UeiDaqAPI Int32 UeiDaq::CUeiDataStream::GetNumberOfScans ()

Get the number of scans to read at a time

Returns:

the number of scans.

Examples:

AnalogInBuffered.cpp, **AnalogInBufferedAsync.cpp**, **AnalogOutBuffered.cpp**, and **AnalogOutBufferedAsync.cpp**.

2.73.2.2 UeiDaqAPI void UeiDaq::CUeiDataStream::SetNumberOfScans (Int32 *numScans*)

Set the number of scans to read at a time

Parameters:

numScans the number of scans.

2.73.2.3 UeiDaqAPI Int32 UeiDaq::CUeiDataStream::GetScanSize ()

Get the size of a scan, usually equal to the number of channels in the session's channel list.

Returns:

the number of samples per scan.

2.73.2.4 UeiDaqAPI void UeiDaq::CUeiDataStream::SetScanSize (Int32 *scanSize*)

Set the size of a scan, usually equal to the number of channels in the session's channel list.

Parameters:

scanSize the number of samples per scan.

2.73.2.5 UeiDaqAPI Int32 UeiDaq::CUeiDataStream::GetSampleSize ()

Get the size of a raw sample in bytes.

Returns:

the number of bytes in one sample.

2.73.2.6 UeiDaqAPI Int32 UeiDaq::CUeiDataStream::GetNumberOfFrames ()

Get the number of frames. Each frame is accessed independently by the device.

Returns:

the number of frames.

2.73.2.7 UeiDaqAPI void UeiDaq::CUeiDataStream::SetNumberOfFrames (Int32 *numFrames*)

Set the number of frames. Each frame is accessed independently by the device.

Parameters:

numFrames the number of frames.

2.73.2.8 UeiDaqAPI Int32 UeiDaq::CUeiDataStream::GetRegenerate ()

Get the parameter that determines whether the output device continuously regenerate the first buffer it received.

Returns:

the regenerate setting value.

2.73.2.9 UeiDaqAPI void UeiDaq::CUeiDataStream::SetRegenerate (Int32 *regen*)

Specifies whether the output device will continuously regenerate the first buffer it received.

Parameters:

regen the regenerate setting value.

2.73.2.10 UeiDaqAPI Int32 UeiDaq::CUeiDataStream::GetBurst ()

Get the parameter that determines whether the input device should acquire all the requested data in its internal memory before transferring it to the host PC.

Returns:

the burst setting value.

2.73.2.11 UeiDaqAPI void UeiDaq::CUeiDataStream::SetBurst (Int32 *burst*)

Specifies whether the input device will acquire all the requested data in its internal memory before transferring it to the host PC.

Parameters:

burst the burst setting value.

2.73.2.12 UeiDaqAPI Int32 UeiDaq::CUeiDataStream::GetOverUnderRun ()

For an input session, Determines whether the device overwrite acquired data if it was not read fast enough. For an output session, Determines whether the device regenerate previously generated data if it didn't receive new data fast enough.

Returns:

the overrun setting value.

2.73.2.13 UeiDaqAPI void UeiDaq::CUeiDataStream::SetOverUnderRun (Int32 *overUnderRun*)

For an input session, Specifies whether the device overwrite acquired data if it was not read fast enough. For an output session, Specifies whether the device regenerate previously generated data if it didn't receive new data fast enough. If overUnderRun is set to 1, the framework ignore the overrun or underrun condition. If it is set to 0 an exception is thrown upon overrun or underrun.

Parameters:

overUnderRun the overrun setting value.

2.73.2.14 UeiDaqAPI bool UeiDaq::CUeiDataStream::IsBlockingEnabled ()

When enabled, read operations will block until the requested amount of data is received from the device. when disabled, read operations return immediately with the amount of data available.

Returns:

the blocking setting value.

2.73.2.15 UeiDaqAPI void UeiDaq::CUeiDataStream::EnableBlocking (bool *blocking*)

When enabled, read operations will block until the requested amount of data is received from the device. when disabled, read operations return immediately with the amount of data available.

Parameters:

blocking the blocking setting value.

2.73.2.16 UeiDaqAPI Int32 UeiDaq::CUeiDataStream::GetCurrentScan ()

Get the latest acquired scan position in the acquisition buffer.

Returns:

the current scan position.

2.73.2.17 UeiDaqAPI Int32 UeiDaq::CUeiDataStream::GetAvailableScans ()

Get the number of scans ready to be read or written

Returns:

the number of available scans.

2.73.2.18 UeiDaqAPI Int32 UeiDaq::CUeiDataStream::GetTotalScans ()

Get the total number of scans read or written since the session started.

Returns:

the total number of scans.

2.73.2.19 UeiDaqAPI void UeiDaq::CUeiDataStream::SetOffset (Int32 *offset*)

Move the position of the next scan to read or write. The offset is relative to the reference specified with **SetRelativeTo()** (p. 182). Use a negative offset to read or write scans below the reference position.

Parameters:

offset the offset.

2.73.2.20 UeiDaqAPI tUeiDataStreamRelativeTo UeiDaq::CUeiDataStream::GetRelativeTo ()

Get the position used as a reference in the framework's internal buffer

Returns:

the reference.

2.73.2.21 UeiDaqAPI void UeiDaq::CUeiDataStream::SetRelativeTo (tUeiDataStreamRelativeTo *relativeTo*)

Set the position to use as a reference in the framework's internal buffer

Parameters:

relativeTo the new reference.

2.73.2.22 UeiDaqAPI bool UeiDaq::CUeiDataStream::IsStreamActive ()

Get the status of the data stream

Returns:

the status

2.73.2.23 UeiDaqAPI void UeiDaq::CUeiDataStream::ReadRawData (Int32 *numScans*, uInt16 * *pBuffer*, IUeiEventListener * *pListener* = NULL)

Read the specified number of raw scans from the stream. This function can work synchronously or asynchronously if a listener interface pointer is provided

Parameters:

numScans Number of scans to read

pBuffer Destination buffer for the scans

pListener Interface pointer for asynchronous callback

2.73.2.24 UeiDaqAPI void UeiDaq::CUeiDataStream::ReadRawData (Int32 *numScans*, uInt32 * *pBuffer*, IUeiEventListener * *pListener* = NULL)

Read the specified number of raw scans from the stream. This function can work synchronously or asynchronously if a listener interface pointer is provided

Parameters:

numScans Number of scans to read

pBuffer Destination buffer for the scans

pListener Interface pointer for asynchronous callback

2.73.2.25 UeiDaqAPI void UeiDaq::CUeiDataStream::ReadCalibratedRawData (Int32 *numScans*, uInt16 * *pBuffer*, IUeiEventListener * *pListener* = NULL)

Read the specified number of calibrated scans from the stream. This function can work synchronously or asynchronously if a listener interface pointer is provided

Parameters:

numScans Number of scans to read

pBuffer Destination buffer for the scans

pListener Interface pointer for asynchronous callback

2.73.2.26 UeiDaqAPI void UeiDaq::CUeiDataStream::ReadCalibratedRawData (Int32 *numScans*, uInt32 * *pBuffer*, IUeiEventListener * *pListener* = NULL)

Read the specified number of calibrated scans from the stream. This function can work synchronously or asynchronously if a listener interface pointer is provided

Parameters:

numScans Number of scans to read

pBuffer Destination buffer for the scans

pListener Interface pointer for asynchronous callback

2.73.2.27 UeiDaqAPI void UeiDaq::CUeiDataStream::ReadScaledData (Int32 *numScans*, f64 * *pBuffer*, IUeiEventListener * *pListener* = NULL)

Read the specified number of scaled scans from the stream. This function can work synchronously or asynchronously if a listener interface pointer is provided

Parameters:

numScans Number of scans to read

pBuffer Destination buffer for the scans

pListener Interface pointer for asynchronous callback

2.73.2.28 UeiDaqAPI void UeiDaq::CUeiDataStream::WriteRawData (Int32 *numScans*, uInt16 * *pBuffer*, IUeiEventListener * *pListener* = NULL)

Write the specified number of raw scans to the stream. This function can work synchronously or asynchronously if a listener interface pointer is provided

Parameters:

numScans Number of scans to write

pBuffer Source buffer for the scans

pListener Interface pointer for asynchronous callback

2.73.2.29 UeiDaqAPI void UeiDaq::CUeiDataStream::WriteRawData (Int32 *numScans*, uInt32 * *pBuffer*, IUeiEventListener * *pListener* = NULL)

Write the specified number of raw scans to the stream. This function can work synchronously or asynchronously if a listener interface pointer is provided

Parameters:

numScans Number of scans to write

pBuffer Source buffer for the scans

pListener Interface pointer for asynchronous callback

2.73.2.30 UeiDaqAPI void UeiDaq::CUeiDataStream::WriteScaledData (Int32 *numScans*, f64 * *pBuffer*, IUeiEventListener * *pListener* = NULL)

Write the specified number of scaled scans to the stream. This function can work synchronously or asynchronously if a listener interface pointer is provided

Parameters:

numScans Number of scans to write

pBuffer Source buffer for the scans

pListener Interface pointer for asynchronous callback

2.73.2.31 UeiDaqAPI void UeiDaq::CUeiDataStream::ReadMessage (Int32 *port*, Int32 *messageType*, Int32 *bufferSize*, Int8 * *pBuffer*, Int32 * *numBytesRead*, IUeiEventListener * *pListener* = NULL)

Read a data message from the stream. The device assiociated with this data stream must support message based operations (such as a serial port or a bus)

Parameters:

port The port to read from

messageType The type of message stored in the buffer

bufferSize Maximum number of bytes that can be stored in the buffer

pBuffer Destination buffer for the message

numBytesRead The actual number of bytes read

pListener Interface pointer for asynchronous callback

2.73.2.32 UeiDaqAPI void UeiDaq::CUeiDataStream::WriteMessage (Int32 *port*, Int32 *messageType*, Int32 *bufferSize*, Int8 * *pBuffer*, Int32 * *numBytesWritten*, IUeiEventListener * *pListener* = NULL)

Write a data message to the stream. The device assiociated with this data stream must support message based operations (such as a serial port or a bus)

Parameters:

port the port to write to

messageType The type of message stored in the buffer

bufferSize Maximum number of bytes that can be stored in the buffer

pBuffer Destination buffer for the message

numBytesWritten The number of actual bytes written

pListener Interface pointer for asynchronous callback

2.73.2.33 UeiDaqAPI void UeiDaq::CUeiDataStream::ReadARINCWords (Int32 *port*, Int32 *bufferSize*, tUeiARINCWord * *pBuffer*, Int32 * *numWordsRead*, IUeiEventListener * *pListener* = NULL)

Read ARINC words from the stream. The device assiociated with this data stream must support ARINC-429

Parameters:

port The port to read from

bufferSize Maximum number of words that can be stored in the buffer

pBuffer Destination buffer for the words

numWordsRead The actual number of words read

pListener Interface pointer for asynchronous callback

2.73.2.34 UeiDaqAPI void UeiDaq::CUeiDataStream::WriteARINCWords (Int32 *port*, Int32 *bufferSize*, tUeiARINCWord * *pBuffer*, Int32 * *numWordsWritten*, IUeiEventListener * *pListener* = NULL)

Write ARINC words to the stream. The device assiociated with this data stream must support ARINC-429

Parameters:

port the port to write to

bufferSize Maximum number of words that can be stored in the buffer

pBuffer Source buffer of the words

numWordsWritten The number of actual Words written

pListener Interface pointer for asynchronous callback

2.73.2.35 UeiDaqAPI void UeiDaq::CUeiDataStream::WriteScheduledARINCWords (Int32 *port*, Int32 *firstWord*, Int32 *bufferSize*, tUeiARINCWord * *pBuffer*, Int32 * *numWordsWritten*, IUeiEventListener * *pListener* = NULL)

Updated scheduled ARINC words with new data. The device assiociated with this data stream must support ARINC-429

Parameters:

port the port to write to

firstWord Index of the first word to update in the scheduler table

bufferSize Maximum number of words that can be stored in the buffer

pBuffer Source buffer of the words

numWordsWritten The number of actual Words written

pListener Interface pointer for asynchronous callback

2.73.2.36 UeiDaqAPI void UeiDaq::CUeiDataStream::WriteScheduledARINCRawWords (Int32 *port*, Int32 *firstWord*, Int32 *bufferSize*, uInt32 * *pBuffer*, Int32 * *numWordsWritten*, IUeiEventListener * *pListener* = NULL)

Updated scheduled ARINC words with new data. The device assiociated with this data stream must support ARINC-429

Parameters:

port the port to write to

firstWord Index of the first word to update in the scheduler table

bufferSize Maximum number of words that can be stored in the buffer

pBuffer Source buffer of the words

numWordsWritten The number of actual Words written

pListener Interface pointer for asynchronous callback

2.73.2.37 UeiDaqAPI void UeiDaq::CUeiDataStream::Flush (Int32 *port*, tUeiFlushAction *action*)

Flush transmit and or receive buffer(s) associated with the specified port

Parameters:

port the port to flush

action the action to be taken while flushing the buffer(s)

2.73.2.38 UeiDaqAPI Int32 UeiDaq::CUeiDataStream::GetAvailableInputMessages (Int32 *port*)

Get the number of messages ready to be read from the specified port

Parameters:

port the port to query

Returns:

the number of available input messages.

2.73.2.39 UeiDaqAPI Int32 UeiDaq::CUeiDataStream::GetAvailableOutputSlots (Int32 *port*)

Get the number of slots ready to accept new messages to the specified port

Parameters:

port the port to query

Returns:

the number of available outputs slots.

2.73.2.40 UeiDaqAPI Int32 UeiDaq::CUeiDataStream::GetTotalMessages (Int32 *port*)

Get the total number of messages read from the specified port since the session started.

Parameters:

port the port to query

Returns:

the total number of messages.

The documentation for this class was generated from the following file:

- UeiDataStream.h

2.74 UeiDaq::CUeiDevice Class Reference

A device object.

`#include <UeiDevice.h>`

Inherits **UeiDaq::CUeiObject**.

Public Member Functions

- UeiDaqAPI **CUeiDevice** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiDevice** ()
  Destructor.
- UeiDaqAPI std::string **GetDeviceName** ()
  Get the device name.
- UeiDaqAPI std::string **GetSerialNumber** ()
  Get the serial number.
- UeiDaqAPI std::string **GetResourceName** ()
  Get the resource descriptor.
- UeiDaqAPI **Int32 GetIndex** ()
  Get the index.
- UeiDaqAPI **Int32 GetNumberOfSubsystems** (**tUeiSessionType** type)
  Get the number of subsystems of the specified type.
- UeiDaqAPI **Int32 GetNumberOfChannels** (**tUeiSessionType** type, int subSytemIndex, int channelGroupIndex=0)
  Get the number of channel of the specified subsystem.
- UeiDaqAPI **Int32 GetMaxRate** (**tUeiSessionType** type, int index)
  Get the maximum rate of the specified subsystem.
- UeiDaqAPI **Int32 GetResolution** (**tUeiSessionType** type, int index)
  Get the resolution of the specified subsystem.
- UeiDaqAPI void **GetRanges** (**tUeiSessionType** type, int index, tRanges &ranges)
  Get the input (or output) ranges of the specified subsystem.
- UeiDaqAPI void **GetGains** (**tUeiSessionType** type, int index, tGains &gains)
  Get the input gains of the specified subsystem.
- UeiDaqAPI **Int32 GetNumberOfAIDifferentialChannels** ()
  Get the number of Analog Input differential channels.

- UeiDaqAPI **Int32 GetNumberOfAISingleEndedChannels** ()
  Get the number of Analog Input single-ended channels.
- UeiDaqAPI **Int32 GetNumberOfAIChannels** (**tUeiAIChannelInputMode** mode)
  Get the number of Analog Input channels.
- UeiDaqAPI **Int32 GetNumberOfAOChannels** ()
  Get the number of Analog Outout channels.
- UeiDaqAPI **Int32 GetNumberOfDIChannels** ()
  Get the number of Digital Input channels.
- UeiDaqAPI **Int32 GetNumberOfDOChannels** ()
  Get the number of Digital Output channels.
- UeiDaqAPI **Int32 GetNumberOfCIChannels** ()
  Get the number of Counter Input channels.
- UeiDaqAPI **Int32 GetNumberOfCOChannels** ()
  Get the number of Counter Output channels.
- UeiDaqAPI **Int32 GetMaxAIRate** ()
  Get the maximum AI rate.
- UeiDaqAPI **Int32 GetMaxAORate** ()
  Get the maximum AO rate.
- UeiDaqAPI **Int32 GetMaxDIRate** ()
  Get the maximum DI rate.
- UeiDaqAPI **Int32 GetMaxDORate** ()
  Get the maximum DO rate.
- UeiDaqAPI **Int32 GetMaxCIRate** ()
  Get the maximum CI rate.
- UeiDaqAPI **Int32 GetMaxCORate** ()
  Get the maximum CO rate.
- UeiDaqAPI **Int32 GetAIResolution** ()
  Get the AI resolution.
- UeiDaqAPI **Int32 GetAOResolution** ()
  Get the AO resolution.
- UeiDaqAPI **Int32 GetDIResolution** ()
  Get the DI resolution.
- UeiDaqAPI **Int32 GetDOResolution** ()
  Get the DO resolution.

- UeiDaqAPI **Int32 GetCIResolution** ()
  Get the CI resolution.
- UeiDaqAPI **Int32 GetCOResolution** ()
  Get the CO resolution.
- UeiDaqAPI **Int32 GetAIDataSize** ()
  Get the AI raw data size.
- UeiDaqAPI **Int32 GetAODataSize** ()
  Get the AO raw data size.
- UeiDaqAPI **Int32 GetDIDataSize** ()
  Get the DI data size.
- UeiDaqAPI **Int32 GetDODataSize** ()
  Get the DO data size.
- UeiDaqAPI **Int32 GetCIDataSize** ()
  Get the CI data size.
- UeiDaqAPI **Int32 GetCODataSize** ()
  Get the CO data size.
- UeiDaqAPI void **GetAIRanges** (tRanges &ranges)
  Get the AI range vector.
- UeiDaqAPI void **GetAORanges** (tRanges &ranges)
  Get the AO range vector.
- UeiDaqAPI void **GetDIRanges** (tRanges &ranges)
  Get the DI range vector.
- UeiDaqAPI void **GetDORanges** (tRanges &ranges)
  Get the DO range vector.
- UeiDaqAPI void **GetAIGains** (tGains &gains)
  Get the AI gain vector.
- UeiDaqAPI bool **IsAISimultaneous** ()
  Check if the device supports simultaneous sampling.
- UeiDaqAPI bool **IsAOSimultaneous** ()
  Check if the device supports simultaneous update.
- UeiDaqAPI **Int32 GetInputFIFOSize** ()
  Get the size of the input FIFO.
- UeiDaqAPI **Int32 GetOutputFIFOSize** ()

Get the size of the output FIFO.

- UeiDaqAPI bool **CanRegenerate** ()
  Check if the device supports output regeneration.
- UeiDaqAPI bool **BidirectionalDigitalPorts** ()
  Check if the device digital ports are bidirectional.
- UeiDaqAPI bool **SupportsDigitalPortHysteresis** ()
  Check if the device supports programmable hysteresis on its digital ports.
- UeiDaqAPI bool **HasFIRFilters** ()
  Check if the device has programmable digital FIR filters.
- UeiDaqAPI **Int32 GetNumberOfSerialPorts** ()
  Get the number of serial ports available on the device.
- UeiDaqAPI **Int32 GetNumberOfSynchronousSerialPorts** ()
  Get the number of synchronous serial ports available on the device.
- UeiDaqAPI **Int32 GetNumberOfCANPorts** ()
  Get the number of CAN ports available on the device.
- UeiDaqAPI **Int32 GetNumberOfARINCInputPorts** ()
  Get the number of ARINC input ports available on the device.
- UeiDaqAPI **Int32 GetNumberOfARINCOutputPorts** ()
  Get the number of ARINC output ports available on the device.
- UeiDaqAPI **Int32 GetNumberOfMIL1553Ports** ()
  Get the number of MIL-1553 ports available on the device.
- UeiDaqAPI **Int32 GetNumberOfIRIGInputPorts** ()
  Get the number of IRIG input ports available on the device.
- UeiDaqAPI **Int32 GetNumberOfIRIGOutputPorts** ()
  Get the number of IRIG output ports available on the device.
- UeiDaqAPI **Int32 GetNumberOfSSIInputPorts** ()
  Get the number of SSI input ports available on the device.
- UeiDaqAPI **Int32 GetNumberOfSSIOutputPorts** ()
  Get the number of SSI output ports available on the device.
- UeiDaqAPI **Int32 GetNumberOfI2CMasterPorts** ()
  Get the number of I2C master ports available on the device.
- UeiDaqAPI **Int32 GetNumberOfI2CSlavePorts** ()
  Get the number of I2C slave ports available on the device.

- UeiDaqAPI **Int32 GetNumberOfDiagnosticChannels** ()
  Get the number of diagnostic channels available on the device.

- UeiDaqAPI bool **SupportsDigitalFiltering** ()
  Check if the device supports digital filtering (debouncing).

- UeiDaqAPI std::string **GetHardwareInformation** ()
  Get hardware information string.

- UeiDaqAPI **Int32 GetSlot** ()
  Get the slot number occupied by the device.

- UeiDaqAPI bool **IsSessionTypeSupported** (**tUeiSessionType** sessionType)
  Checks whether the device support a session type.

- virtual UeiDaqAPI std::string **GetStatus** ()
  Get device status string.

- virtual UeiDaqAPI int **ReadEEPROM** (int bank, int subSystem, **tUeiEEPROMArea** area, **uInt8** *buffer)
  Returns:
  The number of bytes read

- virtual UeiDaqAPI void **WriteEEPROM** (int bank, int subSystem, **tUeiEEPROMArea** area, int size, **uInt8** *buffer, bool saveEEPROM)
  Parameters:
  saveEEPROM *True to commit changes to EEPROM*

- virtual UeiDaqAPI **uInt32 ReadRegister32** (**uInt32** offset)
  Read Device register.

- virtual UeiDaqAPI void **WriteRegister32** (**uInt32** offset, **uInt32** value)
  Write Device register.

- virtual UeiDaqAPI void **WriteReadRegisters32** (**uInt32** writeAddr, **uInt32** writeSize, **uInt32** *writeData, **uInt32** readAddr, **uInt32** readSize, **uInt32** *readData, bool doReadFirst, bool noIncrement)
  Performs a write and read of multiple values into the address space of the Device.

- virtual UeiDaqAPI int **ReadRAM** (**uInt32** address, int size, **uInt8** *buffer)
  Read from device RAM.

- virtual UeiDaqAPI void **WriteRAM** (**uInt32** address, int size, **uInt8** *buffer)
  Write to device RAM.

- UeiDaqAPI **Int32 GetTimeout** ()
  Get the low-level access timeout.

- UeiDaqAPI void **SetTimeout** (**Int32** timeout)
  Set the low-level access timeout.

- virtual UeiDaqAPI void **Reset** (void)
  Reset device.
- virtual UeiDaqAPI void **SetWatchDogCommand** (**tUeiWatchDogCommand** cmd, int timeout)
  Set watchdog command.
- virtual UeiDaqAPI void **WriteCalibration** (int channel, **uInt8** dacMode, **uInt32** value, bool saveEEPROM)
  Write calibration value in device's EEPROM.
- virtual UeiDaqAPI int **Calibrate** ()
  Calibrate.
- virtual UeiDaqAPI int **WriteChannel** (int channel, **uInt32** data)
  WriteChannel.
- virtual UeiDaqAPI int **EnableChannels** ()
  EnableChannels.
- virtual UeiDaqAPI void **ReadInitParameter** (**tUeiInitParameter** parameter, **uInt8** *buffer)
  Read device init parameter.
- virtual UeiDaqAPI void **WriteInitParameter** (**tUeiInitParameter** parameter, **uInt8** *buffer, bool saveInitPrm)
  Write device init parameter.

2.74.1 Detailed Description

Store informations about a device

2.74.2 Member Function Documentation

2.74.2.1 UeiDaqAPI std::string UeiDaq::CUeiDevice::GetDeviceName ()

Get the name from the device database

Returns:

a string that contains the name of the device.

2.74.2.2 UeiDaqAPI std::string UeiDaq::CUeiDevice::GetSerialNumber ()

Get the serial number from the device EEPROM

Returns:

a string that contains the serial number.

2.74.2.3 UeiDaqAPI std::string UeiDaq::CUeiDevice::GetResourceName ()

get the device resource descriptor

Returns:

a string that contains the resource descriptor

2.74.2.4 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetIndex ()

Get the index of this device in the device class

Returns:

the index.

2.74.2.5 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetNumberOfSubsystems (tUeiSessionType *type*)

Get the number of subsystems available on the device

Returns:

the number of subsystems.

2.74.2.6 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetNumberOfChannels (tUeiSessionType *type*, int *subSytemIndex*, int *channelGroupIndex* = 0)

Get the number of channels available on the device

Returns:

the number of channels.

2.74.2.7 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetMaxRate (tUeiSessionType *type*, int *index*)

Get the maximum rate of the specified subsystem on the device

Returns:

the maximum rate

2.74.2.8 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetResolution (tUeiSessionType *type*, int *index*)

Get the resolution of the specified subsystem

Returns:

the resolution.

2.74.2.9 UeiDaqAPI void UeiDaq::CUeiDevice::GetRanges (tUeiSessionType *type*, int *index*, tRanges & *ranges*)

Get the ranges of the specified subsystem

Returns:

the resolution.

2.74.2.10 UeiDaqAPI void UeiDaq::CUeiDevice::GetGains (tUeiSessionType *type*, int *index*, tGains & *gains*)

Get the gains of the specified subsystem

Returns:

the resolution.

2.74.2.11 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetNumberOfAIDifferentialChannels ()

Get the number of Analog Input differential channels

Returns:

the number of AI differential channels.

2.74.2.12 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetNumberOfAISingleEndedChannels ()

Get the number of Analog Input single-ended channels

Returns:

the number of AI single-ended channels.

2.74.2.13 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetNumberOfAIChannels (tUeiAIChannelInputMode *mode*)

Get the number of Analog Input channels available in the specified mode

Parameters:

mode the input mode.

Returns:

the number of AI channels.

2.74.2.14 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetNumberOfAOChannels ()

Get the number of Analog Output channels

Returns:

the number of AO channels.

2.74.2.15 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetNumberOfDIChannels ()

Get the number of Digital Input channels

Returns:

the number of DI channels.

2.74.2.16 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetNumberOfDOChannels ()

Get the number of Digital Output channels

Returns:

the number of DO channels.

2.74.2.17 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetNumberOfCIChannels ()

Get the number of Counter Input channels

Returns:

the number of CI channels.

2.74.2.18 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetNumberOfCOChannels ()

Get the number of Counter Output channels

Returns:

the number of CO channels.

2.74.2.19 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetMaxAIRate ()

Get the maximum rate of the Analog Input subsystem

Returns:

the maximum rate.

2.74.2.20 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetMaxAORate ()

Get the maximum rate of the Analog Output subsystem

Returns:

the maximum rate.

2.74.2.21 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetMaxDIRate ()

Get the maximum rate of the Digital Input subsystem

Returns:

the maximum rate.

2.74.2.22 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetMaxDORate ()

Get the maximum rate of the Digital Output subsystem

Returns:

the maximum rate.

2.74.2.23 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetMaxCIRate ()

Get the maximum rate of the Counter Input subsystem

Returns:

the maximum rate.

2.74.2.24 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetMaxCORate ()

Get the maximum rate of the Counter Output subsystem

Returns:

the maximum rate.

2.74.2.25 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetAIResolution ()

Get the resolution of the Analog Input channels

Returns:

the resolution.

2.74.2.26 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetAOResolution ()

Get the resolution of the Analog Output channels

Returns:

the resolution.

2.74.2.27 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetDIResolution ()

Get the number of lines in a digital input port

Returns:

the resolution.

2.74.2.28 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetDOResolution ()

Get the number of lines in a digital output port

Returns:

the resolution.

2.74.2.29 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetCIResolution ()

Get the input resolution of the counter/timer

Returns:

the resolution.

Examples:

MeasureFrequency.cpp.

2.74.2.30 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetCOResolution ()

Get the output resolution of the counter/timer

Returns:

the resolution.

2.74.2.31 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetAIDataSize ()

Get the number of bytes needed to store each Analog input sample

Returns:

the data size.

2.74.2.32 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetAODataSize ()

Get the number of bytes needed to store each Analog output sample

Returns:

the data size.

2.74.2.33 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetDIDataSize ()

Get the number of bytes needed to store each Digital input sample

Returns:

the data size.

2.74.2.34 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetDODataSize ()

Get the number of bytes needed to store each Digital output sample

Returns:

the data size.

2.74.2.35 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetCIDataSize ()

Get the number of bytes needed to store each Counter input sample

Returns:

the data size.

2.74.2.36 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetCODataSize ()

Get the number of bytes needed to store each Counter output sample

Returns:

the data size.

2.74.2.37 UeiDaqAPI void UeiDaq::CUeiDevice::GetAIRanges (tRanges & *ranges*)

Get the vector containing the Analog input ranges

Parameters:

ranges the range vector

2.74.2.38 UeiDaqAPI void UeiDaq::CUeiDevice::GetAORanges (tRanges & *ranges*)

Get the vector containing the Analog output ranges

Parameters:

ranges the range vector

2.74.2.39 UeiDaqAPI void UeiDaq::CUeiDevice::GetDIRanges (tRanges & *ranges*)

Get the vector containing the Digital input ranges The first member of each pair is the low threshold level and the second member is the high threshold level.

Parameters:

ranges the range vector

2.74.2.40 UeiDaqAPI void UeiDaq::CUeiDevice::GetDORanges (tRanges & *ranges*)

Get the vector containing the Digital output ranges The first member of each pair is the low threshold level and the second member is the high threshold level.

Parameters:

ranges the range vector

2.74.2.41 UeiDaqAPI void UeiDaq::CUeiDevice::GetAIGains (tGains & *gains*)

Get the vector containing the Analog input gains

Parameters:

gains the gain vector

2.74.2.42 UeiDaqAPI bool UeiDaq::CUeiDevice::IsAISimultaneous ()

Check if the device can do simultaneous analog input sampling

Returns:

true if the device supports simulataneous sampling

2.74.2.43 UeiDaqAPI bool UeiDaq::CUeiDevice::IsAOSimultaneous ()

Check if the device can do simultaneous analog output update

Returns:

true if the device supports simulataneous update

2.74.2.44 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetInputFIFOSize ()

Get the size of the input FIFO in number of samples

Returns:

the number of samples that can be stored in the input FIFO

2.74.2.45 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetOutputFIFOSize ()

Get the size of the output FIFO in number of samples

Returns:

the number of samples that can be stored in the output FIFO

2.74.2.46 UeiDaqAPI bool UeiDaq::CUeiDevice::CanRegenerate ()

Check if the device supports output regeneration

Returns:

true if the device supports ouput regeneration

2.74.2.47 UeiDaqAPI bool UeiDaq::CUeiDevice::BidirectionalDigitalPorts ()

Check if the device digital ports are bidirectional

Returns:

true if the device has bidirectional digital ports

2.74.2.48 UeiDaqAPI bool UeiDaq::CUeiDevice::SupportsDigitalPortHysteresis ()

Check if the device supports programmable hysteresis on its digital ports

Returns:

true if the device supports programmable hysteresis

2.74.2.49 UeiDaqAPI bool UeiDaq::CUeiDevice::HasFIRFilters ()

Check if the device has programmable digital FIR filters

Returns:

true if the device has digital FIR filters

2.74.2.50 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetNumberOfSerialPorts ()

Get the number of serial ports available on the device

Returns:

The number of serial ports

2.74.2.51 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetNumberOfSynchronousSerialPorts ()

Get the number of synchronous serial ports available on the device

Returns:

The number of serial ports

2.74.2.52 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetNumberOfCANPorts ()

Get the number of CAN ports available on the device

Returns:

The number of serial ports

2.74.2.53 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetNumberOfARINCInputPorts ()

Get the number of ARINC input ports available on the device

Returns:

The number of ARINC input ports

2.74.2.54 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetNumberOfARINCOutputPorts ()

Get the number of ARINC output ports available on the device

Returns:

The number of ARINC output ports

2.74.2.55 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetNumberOfMIL1553Ports ()

Get the number of MIL-1553 ports available on the device

Returns:

The number of MIL-1553 ports

2.74.2.56 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetNumberOfIRIGInputPorts ()

Get the number of IRIG input ports available on the device

Returns:

The number of IRIG input ports

2.74.2.57 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetNumberOfIRIGOutputPorts ()

Get the number of IRIG output ports available on the device

Returns:

The number of IRIG output ports

2.74.2.58 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetNumberOfSSIInputPorts ()

Get the number of SSI input ports available on the device

Returns:

The number of SSI input ports

2.74.2.59 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetNumberOfSSIOutputPorts ()

Get the number of SSI output ports available on the device

Returns:

The number of SSI output ports

2.74.2.60 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetNumberOfI2CMasterPorts ()

Get the number of I2C master ports available on the device

Returns:

The number of I2C master ports

2.74.2.61 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetNumberOfI2CSlavePorts ()

Get the number of I2C slave ports available on the device

Returns:

The number of I2C slave ports

2.74.2.62 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetNumberOfDiagnosticChannels ()

Get the number of diagnostic channels available on the device

Returns:

The number of diagnostic channels

2.74.2.63 UeiDaqAPI bool UeiDaq::CUeiDevice::SupportsDigitalFiltering ()

Check if the device supports digital filtering on its Counter/timer inputs

Returns:

true if the device has digital filters

2.74.2.64 UeiDaqAPI std::string UeiDaq::CUeiDevice::GetHardwareInformation ()

Get low-level hardware information string. It contains informations such as firmware revision, logic revision, base address and interrupt configuration.

Returns:

A string containing hardware informations about the device.

2.74.2.65 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetSlot ()

Get the slot number occupied by the device

Returns:

The slot device is inserted in

2.74.2.66 UeiDaqAPI bool UeiDaq::CUeiDevice::IsSessionTypeSupported (tUeiSessionType *sessionType*)

Checks whether the device supports the specified session type

Parameters:

sessionType the session type

Returns:

true if the session type is supported, false otherwise

2.74.2.67 virtual UeiDaqAPI std::string UeiDaq::CUeiDevice::GetStatus () [virtual]

Read the device status string, the status string contains status information specific to each device. It is for internal use only.

Returns:

A string containing the of status the device.

2.74.2.68 virtual UeiDaqAPI int UeiDaq::CUeiDevice::ReadEEPROM (int *bank*, int *subSystem*, tUeiEEPROMArea *area*, uInt8 * *buffer*) [virtual]

Read the device EEPROM content. EEPROM contains informations such as Manufacture date, calibration date and calibration data The EEPROM content is returned as an opaque array of bytes. Each device has a unique way of storing informations in its EEPROM and it is up to the calling application to interpret its content.

Some devices have multiple EEPROM areas. Use the bank parameter to specify which EEPROM you wish to read from

Parameters:

bank The EEPROM bank to read from

subSystem The sub-system to read EEPROM from

area The area to access in the EEPROM

buffer Buffer big enough to store the EEPROM content

2.74.2.69 virtual UeiDaqAPI void UeiDaq::CUeiDevice::WriteEEPROM (int *bank*, int *subSystem*, tUeiEEPROMArea *area*, int *size*, uInt8 * *buffer*, bool *saveEEPROM*) [virtual]

Write the device EEPROM content. EEPROM contains informations such as Manufacture date, calibration date and calibration data The EEPROM content is sent as an opaque array of bytes. Each device has a unique way of storing informations in its EEPROM and it is up to the calling application to prepare the buffer to match the EEPROM structure.

Some devices have multiple EEPROM areas. Use the bank parameter to specify which EEPROM you wish to write to

Parameters:

bank The EEPROM bank to write to

subSystem The sub-system to write to

area The area to access in the EEPROM

size The number of bytes to write

buffer Buffer containing the new EEPROM content

2.74.2.70 virtual UeiDaqAPI uInt32 UeiDaq::CUeiDevice::ReadRegister32 (uInt32 *offset*) [virtual]

Read a device register as a 32 bits value.

Parameters:

offset Offset of the register relative to device's base address

Returns:

The register content

2.74.2.71 virtual UeiDaqAPI void UeiDaq::CUeiDevice::WriteRegister32 (uInt32 *offset*, uInt32 *value*) [virtual]

Write a device register as a 32 bits value.

Parameters:

offset Offset of the register relative to device's base address

value The value to write to the register

2.74.2.72 virtual UeiDaqAPI void UeiDaq::CUeiDevice::WriteReadRegisters32 (uInt32 *writeAddr*, uInt32 *writeSize*, uInt32 * *writeData*, uInt32 *readAddr*, uInt32 *readSize*, uInt32 * *readData*, bool *doReadFirst*, bool *noIncrement*) [virtual]

Parameters:

writeAddr Start of address offset that will be written to. Length determined by writeData.Length.

writeSize Number of 32-bit values to write to Device.

writeData Buffer containing data to be written, up to 361 elements.

readAddr Start of address offset that will be read from.

readSize Number of 32-bit values to read from Device.

readData Buffer to store data read from the device, up to 361 elements.

doReadFirst Perform the read before the write.

noIncrement Does not increment to next address (useful for FIFOs).

2.74.2.73 virtual UeiDaqAPI int UeiDaq::CUeiDevice::ReadRAM (uInt32 *address*, int *size*, uInt8 * *buffer*) [virtual]

Read from the device RAM. Only for devices that actually have RAM.

Parameters:

address address of the first element to read

buffer Buffer to store the RAM content

size The number of bytes to read from RAM

Returns:

The number of bytes read

2.74.2.74 virtual UeiDaqAPI void UeiDaq::CUeiDevice::WriteRAM (uInt32 *address*, int *size*, uInt8 * *buffer*) [virtual]

Write to the device RAM. Only for devices that actually have RAM.

Parameters:

address address of the first element to write

size The number of bytes to write

buffer Buffer containing the new value to store in RAM

2.74.2.75 UeiDaqAPI Int32 UeiDaq::CUeiDevice::GetTimeout ()

Get the maximum amount of time (in ms) for a low-level access request to complete.

Returns:

the timeout

See also:

SetTimeout (p. 207)

2.74.2.76 UeiDaqAPI void UeiDaq::CUeiDevice::SetTimeout (Int32 *timeout*)

Set the maximum amount of time (in ms) for a low-level access request to complete. Set to -1 to use default value (depends on hardware)

Parameters:

timeout the timeout

See also:

GetTimeout (p. 207)

2.74.2.77 virtual UeiDaqAPI void UeiDaq::CUeiDevice::Reset (void) [virtual]

Executes a hardware reset on the device. To reboot a PowerDNA or PowerDNR unit call this method on the CPU device (device 14).

2.74.2.78 virtual UeiDaqAPI void UeiDaq::CUeiDevice::SetWatchDogCommand (tUeiWatchDogCommand *cmd*, int *timeout*) [virtual]

Enable/Disable watchdog and configure mode. This is only supported on PowerDNA/DNR/DNF CPU devices (device 14)

Parameters:

cmd the watchdog command

timeout the watchdog expiration timeout

2.74.2.79 virtual UeiDaqAPI void UeiDaq::CUeiDevice::WriteCalibration (int *channel*, uInt8 *dacMode*, uInt32 *value*, bool *saveEEPROM*) [virtual]

Write/Replace calibration value in the EEPROM area reserved for the specified channel

Parameters:

channel The channel

dacMode The DAC mode

value The new calibration value

saveEEPROM True to commit all changes to EEPROM

2.74.2.80 virtual UeiDaqAPI int UeiDaq::CUeiDevice::Calibrate () [virtual]

Run device low level self-calibration if implemented It is for internal use only.

Returns:

An int indicating status on return. Device specific.

2.74.2.81 virtual UeiDaqAPI int UeiDaq::CUeiDevice::WriteChannel (int *channel*, uInt32 *data*) [virtual]

Write data to channel. It is for internal use only.

Parameters:

channel The channel

data The data to write to channel

Returns:

An int indicating status on return. Device specific.

2.74.2.82 virtual UeiDaqAPI int UeiDaq::CUeiDevice::EnableChannels () [virtual]

Enable channels on device. It is for internal use only.

Returns:

An int indicating status on return. Device specific.

2.74.2.83 virtual UeiDaqAPI void UeiDaq::CUeiDevice::ReadInitParameter (tUeiInitParameter *parameter*, uInt8 * *buffer*) [virtual]

Read the device init parameter content.

Parameters:

parameter The parameter to read

buffer Buffer big enough to store the init parameter

2.74.2.84 virtual UeiDaqAPI void UeiDaq::CUeiDevice::WriteInitParameter (tUeiInitParameter *parameter*, uInt8 * *buffer*, bool *saveInitPrm*) [virtual]

Write to the device init parameter.

Parameters:

parameter The init parameter to write to

buffer Buffer containing data to write to the init parameter

saveInitPrm True to store changes of parameter changes for parameters stored in flash

The documentation for this class was generated from the following file:

- UeiDevice.h

2.75 UeiDaq::CUeiDeviceEnumerator Class Reference

Device enumerator.

`#include <UeiDeviceEnumerator.h>`

Public Member Functions

- UeiDaqAPI **CUeiDeviceEnumerator** (std::string deviceClass, std::string remoteAddress, unsigned int remotePort)

  Constructor.

- UeiDaqAPI **CUeiDeviceEnumerator** (std::string driverResource)

  Constructor.

- virtual UeiDaqAPI ~**CUeiDeviceEnumerator** ()

  Destructor.

- UeiDaqAPI **CUeiDevice** * **GetDevice** (int deviceIndex)

  Get a device object from its index.

- UeiDaqAPI int **GetNumberOfDevices** ()

  Get the number of devices.

Static Public Member Functions

- static UeiDaqAPI **CUeiDevice** * **GetDeviceFromResource** (std::string resource)

  Get a device object from its resource string.

2.75.1 Detailed Description

Enumerates devices detected on the specified resource

2.75.2 Constructor & Destructor Documentation

2.75.2.1 UeiDaqAPI UeiDaq::CUeiDeviceEnumerator::CUeiDeviceEnumerator (std::string *deviceClass*, std::string *remoteAddress*, unsigned int *remotePort*)

Build a device enumerator from the device class and the remote address strings.

Parameters:

deviceClass the class of device to enumerate (ex: simu, pwrdaq).

remoteAddress the address of the module where the enumeration takes place.

remotePort the port used by the remote module

See also:

GetDevice (p. 211)

2.75.2.2 UeiDaqAPI UeiDaq::CUeiDeviceEnumerator::CUeiDeviceEnumerator (std::string *driverResource*)

Build a device enumerator from the driver resource string (ex: pdna://192.168.0.12/).

Parameters:

driverResource the resource name of the device family to enumerate (ex: simu://, pwrdaq://, pdna://192.168.100.2).

See also:

GetDevice (p. 211)

2.75.3 Member Function Documentation

2.75.3.1 UeiDaqAPI CUeiDevice* UeiDaq::CUeiDeviceEnumerator::GetDevice (int *deviceIndex*)

Enumerates a device.

Returns:

a pointer to the device object or NULL if there is no more devices to enumerate.

See also:

GetNumberOfDevices (p. 211)

2.75.3.2 UeiDaqAPI int UeiDaq::CUeiDeviceEnumerator::GetNumberOfDevices ()

Get the number of enumerated devices

Returns:

the number of enumerated devices.

See also:

GetDevice (p. 211)

2.75.3.3 static UeiDaqAPI CUeiDevice* UeiDaq::CUeiDeviceEnumerator::GetDeviceFromResource (std::string *resource*) [static]

Get a device object pointer specified by its resource string

Returns:

a pointer to the device object or NULL if the device doesn't exist

The documentation for this class was generated from the following file:

- UeiDeviceEnumerator.h

2.76 UeiDaq::CUeiDiagnosticChannel Class Reference

Manages settings for each diagnostic input channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Public Member Functions

- UeiDaqAPI **CUeiDiagnosticChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiDiagnosticChannel** ()
  Destructor.
- UeiDaqAPI **f64 GetWorkableMinimum** ()
  Get channel workable minimum value.
- UeiDaqAPI void **SetWorkableMinimum** (**f64** value)
  Set channel workable minimum value.
- UeiDaqAPI **f64 GetWorkableMaximum** ()
  Get channel workable maximum value.
- UeiDaqAPI void **SetWorkableMaximum** (**f64** value)
  Set channel workable maximum value.
- UeiDaqAPI **f64 GetIdealMinimum** ()
  Get channel ideal minimum value.
- UeiDaqAPI void **SetIdealMinimum** (**f64** value)
  Set channel ideal minimum value.
- UeiDaqAPI **f64 GetIdealMaximum** ()
  Get channel ideal maximum value.
- UeiDaqAPI void **SetIdealMaximum** (**f64** value)
  Set channel ideal maximum value.
- UeiDaqAPI int **GetMovingAverageWindowSize** ()
  Get moving average window size.
- UeiDaqAPI void **SetMovingAverageWindowSize** (int windowSize)
  Set moving average window size.

2.76.1 Detailed Description

Manages settings for each diagnostic input channel

2.76.2 Member Function Documentation

2.76.2.1 UeiDaqAPI int UeiDaq::CUeiDiagnosticChannel::GetMovingAverageWindowSize ()

Get the moving average windows size for devices that have this capability The device needs to be capable of averaging measurement in hardware

Returns:

the current moving average window size

2.76.2.2 UeiDaqAPI void UeiDaq::CUeiDiagnosticChannel::SetMovingAverageWindowSize (int *windowSize*)

Set the moving average windows size for devices that have this capability This is the number of data points used to calculate the average The device needs to be capable of averaging measurements in hardware

Parameters:

windowSize the new moving average window size

The documentation for this class was generated from the following file:

- UeiChannel.h

2.77 UeiDaq::CUeiDialog Class Reference

`#include <UeiDialog.h>`

Static Public Member Functions

- static UeiDaqUiAPI bool **ManageSessions** (HWND hParentWindow, std::string sessionGroup)
- static UeiDaqUiAPI std::string **SelectResource** (HWND hParentWindow, **tUeiSessionType** sessionType)
- static UeiDaqUiAPI std::string **ConfigureSession** (HWND hParentWindow, bool bConfigSessionType=true, bool bConfigTiming=true, bool bConfigTriggers=true, std::string xmlSession="")
- static UeiDaqUiAPI bool **ConfigureSessionFile** (HWND hParentWindow, bool bConfigSessionType, bool bConfigTiming, bool bConfigTriggers, std::string sessionFile)

2.77.1 Detailed Description

This class contains static functions to operate various configuration dialog boxes

2.77.2 Member Function Documentation

2.77.2.1 static UeiDaqUiAPI bool UeiDaq::CUeiDialog::ManageSessions (HWND *hParentWindow*, std::string *sessionGroup*) [static]

Open the sesion manager dialog box

Open the sesion manager dialog box and let the user create a new session and remove or edit an existing session.

Parameters:

hParentWindow The handle of the parent window of the dialog box, if NULL the parent window is the desktop

sessionGroup The group of sessions to edit

Returns:

true if the user validated his changes, false if canceled.

2.77.2.2 static UeiDaqUiAPI std::string UeiDaq::CUeiDialog::SelectResource (HWND *hParentWindow*, tUeiSessionType *sessionType*) [static]

Open the device browser dialog box

Open the device browser dialog box and let the user select a device capable of being used in a session of the specified type

Parameters:

hParentWindow The handle of the parent window of the dialog box, if NULL the parent window is the desktop

sessionType The type of session to be used with the selected device

Returns:

A string that contains the resource of the selected device.

2.77.2.3 static UeiDaqUiAPI std::string UeiDaq::CUeiDialog::ConfigureSession (HWND *hParentWindow*, bool *bConfigSessionType* = true, bool *bConfigTiming* = true, bool *bConfigTriggers* = true, std::string *xmlSession* = "") [static]

Open the session configurator dialog box

Open the session configurator dialog box and let the user configure a session

Parameters:

hParentWindow The handle of the parent window of the dialog box, if NULL the parent window is the desktop

bConfigSessionType Allow the user to change the session type (AI, AO...) in the dialog

bConfigTiming Allow the user to change timing parameters in the dialog

bConfigTriggers Allow the user to change triggers parameters in the dialog

xmlSession A string containing an XL representation of the session parameters.

Returns:

The new configured XML session if the user clicked on "Ok". This string is empty if the user clicked on "Cancel"

2.77.2.4 static UeiDaqUiAPI bool UeiDaq::CUeiDialog::ConfigureSessionFile (HWND *hParentWindow*, bool *bConfigSessionType*, bool *bConfigTiming*, bool *bConfigTriggers*, std::string *sessionFile*) [static]

Open the session configurator dialog box

Open the session configurator dialog box and let the user configure a session stored in a file

Parameters:

hParentWindow The handle of the parent window of the dialog box, if NULL the parent window is the desktop

bConfigSessionType Allow the user to change the session type (AI, AO...) in the dialog

bConfigTiming Allow the user to change timing parameters in the dialog

bConfigTriggers Allow the user to change triggers parameters in the dialog

sessionFile A string containing the name of the session file to configure.

Returns:

true the user clicked on "Ok". false if the user clicked on "Cancel"

The documentation for this class was generated from the following file:

- UeiDialog.h

2.78 UeiDaq::CUeiDIChannel Class Reference

Manages settings for each digital input channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Inherited by **UeiDaq::CUeiDIIndustrialChannel**.

Public Member Functions

- UeiDaqAPI **CUeiDIChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiDIChannel** ()
  Destructor.
- UeiDaqAPI **uInt32 GetEdgeMask** (**tUeiDigitalEdge** edgeType=UeiDigitalEdgeRising)
  Get the edge detection mask.
- UeiDaqAPI void **SetEdgeMask** (**uInt32** mask, **tUeiDigitalEdge** edgeType=UeiDigitalEdgeRising)
  Set the edge detection mask.

2.78.1 Detailed Description

Manages settings for each digital input channel

Examples:

DigitalInEvent.cpp.

2.78.2 Member Function Documentation

2.78.2.1 UeiDaqAPI uInt32 UeiDaq::CUeiDIChannel::GetEdgeMask (tUeiDigitalEdge *edgeType* = `UeiDigitalEdgeRising`)

Get the mask used to select wich digital line will be used to sense edges. When bit x is set to 1, line x is used.

Parameters:

edgeType the edge type to detect

Returns:

the edge detection mask.

See also:

SetEdgeMask (p. 218)

2.78.2.2 UeiDaqAPI void UeiDaq::CUeiDIChannel::SetEdgeMask (uInt32 *mask*, tUeiDigitalEdge *edgeType* = `UeiDigitalEdgeRising`)

Set the mask used to select wich digital line will be used to sense edges. When bit x is set to 1, line x is used.

Parameters:

mask the edge detection mask.

edgeType the edge type to detect

See also:

GetEdgeMask (p. 217)

Examples:

DigitalInEvent.cpp.

The documentation for this class was generated from the following file:

- UeiChannel.h

2.79 UeiDaq::CUeiDigitalReader Class Reference

Digital Reader class.

`#include <UeiReader.h>`

Public Member Functions

- UeiDaqAPI **CUeiDigitalReader** (**CUeiDataStream** *pDataStream)
  Constructor.
- virtual UeiDaqAPI ~**CUeiDigitalReader** ()
  Destructor.
- UeiDaqAPI void **ReadSingleScan** (**uInt16** *pBuffer)
  Read a 16 bits wide scan.
- UeiDaqAPI void **ReadSingleScan** (**uInt32** *pBuffer)
  Read a 32 bits wide scan.
- UeiDaqAPI void **ReadMultipleScans** (**Int32** numScans, **uInt16** *pBuffer)
  Read multiple 16 bits wide scans.
- UeiDaqAPI void **ReadMultipleScans** (**Int32** numScans, **uInt32** *pBuffer)
  Read multiple 32 bits wide scans.
- UeiDaqAPI void **ReadSingleScanAsync** (**uInt16** *pBuffer)
  Read a 16 bits wide scan asynchronously.
- UeiDaqAPI void **ReadSingleScanAsync** (**uInt32** *pBuffer)
  Read a 32 bits wide scan asynchronously.
- UeiDaqAPI void **ReadMultipleScansAsync** (**Int32** numScans, **uInt16** *pBuffer)
  Read multiple 16 bits wide scans asynchronously.
- UeiDaqAPI void **ReadMultipleScansAsync** (**Int32** numScans, **uInt32** *pBuffer)
  Read multiple 32 bits wide scans asynchronously.
- UeiDaqAPI void **AddEventListener** (**IUeiEventListener** *pListener)
  Add an asynchronous listener.

2.79.1 Detailed Description

Class that handles reading data of a stream coming from a digital input

Examples:

DigitalInEvent.cpp.

2.79.2 Constructor & Destructor Documentation

2.79.2.1 UeiDaqAPI UeiDaq::CUeiDigitalReader::CUeiDigitalReader (CUeiDataStream * *pDataStream*)

Parameters:

pDataStream represents the source of data to read

2.79.3 Member Function Documentation

2.79.3.1 UeiDaqAPI void UeiDaq::CUeiDigitalReader::ReadSingleScan (uInt16 * *pBuffer*)

Read only one scan from the input stream

Parameters:

pBuffer destination buffer

2.79.3.2 UeiDaqAPI void UeiDaq::CUeiDigitalReader::ReadSingleScan (uInt32 * *pBuffer*)

Read only one scan from the input stream

Parameters:

pBuffer destination buffer

2.79.3.3 UeiDaqAPI void UeiDaq::CUeiDigitalReader::ReadMultipleScans (Int32 *numScans*, uInt16 * *pBuffer*)

Read several scans from the input stream

Parameters:

numScans number of scans to read

pBuffer destination buffer

2.79.3.4 UeiDaqAPI void UeiDaq::CUeiDigitalReader::ReadMultipleScans (Int32 *numScans*, uInt32 * *pBuffer*)

Read several scans from the input stream

Parameters:

numScans number of scans to read

pBuffer destination buffer

2.79.3.5 UeiDaqAPI void UeiDaq::CUeiDigitalReader::ReadSingleScanAsync (uInt16 * *pBuffer*)

Read only one scan asynchronously from the input stream. The destination buffer contains valid data once the listener is called.

Parameters:

pBuffer destination buffer

2.79.3.6 UeiDaqAPI void UeiDaq::CUeiDigitalReader::ReadSingleScanAsync (uInt32 * *pBuffer*)

Read only one scan asynchronously from the input stream. The destination buffer contains valid data once the listener is called.

Parameters:

pBuffer destination buffer

2.79.3.7 UeiDaqAPI void UeiDaq::CUeiDigitalReader::ReadMultipleScansAsync (Int32 *numScans*, uInt16 * *pBuffer*)

Read several scans asynchronously from the input stream. The destination buffer contains valid data once the listener is called.

Parameters:

numScans number of scans to read

pBuffer destination buffer

2.79.3.8 UeiDaqAPI void UeiDaq::CUeiDigitalReader::ReadMultipleScansAsync (Int32 *numScans*, uInt32 * *pBuffer*)

Read several scans asynchronously from the input stream. The destination buffer contains valid data once the listener is called.

Parameters:

numScans number of scans to read

pBuffer destination buffer

2.79.3.9 UeiDaqAPI void UeiDaq::CUeiDigitalReader::AddEventListener (IUeiEventListener * *pListener*)

Subscribe a listener to receive asynchronous events

Parameters:

pListener pointer to a class that implements **IUeiEventListener** (p. 549) interface

The documentation for this class was generated from the following file:

- UeiReader.h

2.80 UeiDaq::CUeiDigitalWriter Class Reference

Digital Writer class.

`#include <UeiWriter.h>`

Public Member Functions

- UeiDaqAPI **CUeiDigitalWriter** (**CUeiDataStream** *pDataStream)
  Constructor.
- virtual UeiDaqAPI ~**CUeiDigitalWriter** ()
  Destructor.
- UeiDaqAPI void **WriteSingleScan** (**uInt16** *pBuffer)
  Write a 16 bits wide scan.
- UeiDaqAPI void **WriteSingleScan** (**uInt32** *pBuffer)
  Write a 32 bits wide scan.
- UeiDaqAPI void **WriteMultipleScans** (**Int32** numScans, **uInt16** *pBuffer)
  Write multiple 16 bits wide scans.
- UeiDaqAPI void **WriteMultipleScans** (**Int32** numScans, **uInt32** *pBuffer)
  Write multiple 32 bits wide scans.
- UeiDaqAPI void **WriteSingleScanAsync** (**uInt16** *pBuffer)
  Write a 16 bits wide scan asynchronously.
- UeiDaqAPI void **WriteSingleScanAsync** (**uInt32** *pBuffer)
  Write a 32 bits wide scan asynchronously.
- UeiDaqAPI void **WriteMultipleScansAsync** (**Int32** numScans, **uInt16** *pBuffer)
  Write multiple 16 bits wide scans asynchronously.
- UeiDaqAPI void **WriteMultipleScansAsync** (**Int32** numScans, **uInt32** *pBuffer)
  Write multiple 32 bits wide scans asynchronously.
- UeiDaqAPI void **AddEventListener** (**IUeiEventListener** *pListener)
  Add an asynchronous listener.

2.80.1 Detailed Description

Class that handles writing digital data to a stream going to a digital output

2.80.2 Constructor & Destructor Documentation

2.80.2.1 UeiDaqAPI UeiDaq::CUeiDigitalWriter::CUeiDigitalWriter (CUeiDataStream * *pDataStream*)

Parameters:

pDataStream represents the destination where to write data

2.80.3 Member Function Documentation

2.80.3.1 UeiDaqAPI void UeiDaq::CUeiDigitalWriter::WriteSingleScan (uInt16 * *pBuffer*)

Write only one scan to the output stream

Parameters:

pBuffer source buffer

2.80.3.2 UeiDaqAPI void UeiDaq::CUeiDigitalWriter::WriteSingleScan (uInt32 * *pBuffer*)

Write only one scan to the output stream

Parameters:

pBuffer source buffer

2.80.3.3 UeiDaqAPI void UeiDaq::CUeiDigitalWriter::WriteMultipleScans (Int32 *numScans*, uInt16 * *pBuffer*)

Write several scans to the output stream

Parameters:

numScans number of scans to write

pBuffer source buffer

2.80.3.4 UeiDaqAPI void UeiDaq::CUeiDigitalWriter::WriteMultipleScans (Int32 *numScans*, uInt32 * *pBuffer*)

Write several scans to the output stream

Parameters:

numScans number of scans to write

pBuffer source buffer

2.80.3.5 UeiDaqAPI void UeiDaq::CUeiDigitalWriter::WriteSingleScanAsync (uInt16 * *pBuffer*)

Write only one scan asynchronously to the output stream. The source buffer can be reused once the listener is called.

Parameters:

pBuffer source buffer

2.80.3.6 UeiDaqAPI void UeiDaq::CUeiDigitalWriter::WriteSingleScanAsync (uInt32 * *pBuffer*)

Write only one scan asynchronously to the output stream. The source buffer can be reused once the listener is called.

Parameters:

pBuffer source buffer

2.80.3.7 UeiDaqAPI void UeiDaq::CUeiDigitalWriter::WriteMultipleScansAsync (Int32 *numScans*, uInt16 * *pBuffer*)

Write several scans asynchronously to the output stream. The source buffer can be reused once the listener is called.

Parameters:

numScans number of scans to write
pBuffer source buffer

2.80.3.8 UeiDaqAPI void UeiDaq::CUeiDigitalWriter::WriteMultipleScansAsync (Int32 *numScans*, uInt32 * *pBuffer*)

Write several scans asynchronously to the output stream. The source buffer can be reused once the listener is called.

Parameters:

numScans number of scans to write
pBuffer source buffer

2.80.3.9 UeiDaqAPI void UeiDaq::CUeiDigitalWriter::AddEventListener (IUeiEventListener * *pListener*)

Subscribe a listener to receive asynchronous events

Parameters:

pListener pointer to a class that implements **IUeiEventListener** (p. 549) interface

The documentation for this class was generated from the following file:

- UeiWriter.h

2.81 UeiDaq::CUeiDIIndustrialChannel Class Reference

Manages settings for each industrial digital input channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiDIChannel**.

Public Member Functions

- UeiDaqAPI **CUeiDIIndustrialChannel** (int numLines)
  Constructor.
- virtual UeiDaqAPI ~**CUeiDIIndustrialChannel** ()
  Destructor.
- UeiDaqAPI double **GetMinimumPulseWidth** (int line)
  Get the digital input filter minimum pulse width.
- UeiDaqAPI void **SetMinimumPulseWidth** (int line, double minWidth)
  Set the digital input filter minimum pulse width.
- UeiDaqAPI double **GetLowThreshold** (int line)
  Get the low input threshold.
- UeiDaqAPI void **SetLowThreshold** (int line, double lowThreshold)
  Set the low input threshold.
- UeiDaqAPI double **GetHighThreshold** (int line)
  Get the high input threshold.
- UeiDaqAPI void **SetHighThreshold** (int line, double highThreshold)
  Set the high input threshold.
- UeiDaqAPI bool **IsACModeEnabled** (int line)
  Get AC mode.
- UeiDaqAPI void **EnableACMode** (int line, bool acMode)
  Enable or disable AC mode.
- UeiDaqAPI **tUeiDigitalInputMux GetMux** (int line)
  Get the digital input mux configuration.
- UeiDaqAPI void **SetMux** (int line, **tUeiDigitalInputMux** mux)
  Set the digital input mux configuration.
- UeiDaqAPI double **GetVoltageSupply** (int line)
  Get the test/pull-up voltage.
- UeiDaqAPI void **SetVoltageSupply** (int line, double voltage)

Set the test/pull-up voltage.

- UeiDaqAPI int **GetInputGain** (int line)
  Get the input gain.
- UeiDaqAPI void **SetInputGain** (int line, int inputGain)
  Set the input gain.
- UeiDaqAPI int **GetMuxDelay** (int line)
  Get the mux delay.
- UeiDaqAPI void **SetMuxDelay** (int line, int muxDelayUs)
  Set the mux delay.

2.81.1 Detailed Description

Manages settings for each digital input channel

2.81.2 Member Function Documentation

2.81.2.1 UeiDaqAPI double UeiDaq::CUeiDIIndustrialChannel::GetMinimumPulseWidth (int *line*)

The digital input filter is used to block digital noise from switching the input line state. It filters glitches or spikes based on their width.

Parameters:

line The input line to query

Returns:

the minimum pulse width in ms

See also:

SetMinimumPulseWidth (p. 226)

2.81.2.2 UeiDaqAPI void UeiDaq::CUeiDIIndustrialChannel::SetMinimumPulseWidth (int *line*, double *minWidth*)

The digital input filter is used to block digital noise from switching the input line state. It filters glitches or spikes based on their width.

Parameters:

line The input line to configure

minWidth the minimum pulse width in ms. Use 0.0 to disable digital input filter.

See also:

GetMinimumPulseWidth (p. 226)

2.81.2.3 UeiDaqAPI double UeiDaq::CUeiDIIndustrialChannel::GetLowThreshold (int *line*)

The low input threshold is used in combination with the high input threshold to define an hysteresis window. The input line state will change only when the input signal crosses both threshold.

Parameters:

line The input line to query

Returns:

the low threshold

See also:

SetLowThreshold (p. 227)

2.81.2.4 UeiDaqAPI void UeiDaq::CUeiDIIndustrialChannel::SetLowThreshold (int *line*, double *lowThreshold*)

The low input threshold is used in combination with the high input threshold to define an hysteresis window. The input line state will change only when the input signal crosses both threshold

Parameters:

line The input line to configure

lowThreshold the low hysteresis threshold

See also:

GetLowThreshold (p. 227)

2.81.2.5 UeiDaqAPI double UeiDaq::CUeiDIIndustrialChannel::GetHighThreshold (int *line*)

The high input threshold is used in combination with the low input threshold to define an hysteresis window. The input line state will change only when the input signal crosses both threshold.

Parameters:

line The input line to query

Returns:

the high threshold

See also:

SetHighThreshold (p. 228)

2.81.2.6 UeiDaqAPI void UeiDaq::CUeiDIIndustrialChannel::SetHighThreshold (int *line*, double *highThreshold*)

The high input threshold is used in combination with the low input threshold to define an hysteresis window. The input line state will change only when the input signal crosses both threshold

Parameters:

line The input line to configure

highThreshold the high hysteresis threshold

See also:

GetHighThreshold (p. 227)

2.81.2.7 UeiDaqAPI bool UeiDaq::CUeiDIIndustrialChannel::IsACModeEnabled (int *line*)

Verify whether AC mode is enabled or disabled In AC mode, the RMS voltage measured at each input is compared with low and high thresholds to determine the state of the input

Parameters:

line The input line to query

Returns:

the AC mode status

2.81.2.8 UeiDaqAPI void UeiDaq::CUeiDIIndustrialChannel::EnableACMode (int *line*, bool *acMode*)

Enable or disable AC mode In AC mode, the RMS voltage measured at each input is compared with low and high thresholds to determine the state of the input

Parameters:

line The input line to configure

acMode true to enable AC mode, false otherwise

2.81.2.9 UeiDaqAPI tUeiDigitalInputMux UeiDaq::CUeiDIIndustrialChannel::GetMux (int *line*)

Select how voltage supply is connected to each input line. Disconnected: Set the mux to tri-state mode to disconnect voltage supply from input line Diag mode: Set mux to Diagnostic mode to connect internal voltage source to each input line in order to test that it is functional Pull-up mode: Setting mux to pull-up mode connects a pull-up resistor between the internal voltage source and the input line to monitor switches or contacts without external circuitry.

Parameters:

line The input line to query

Returns:

the mux state

2.81.2.10 UeiDaqAPI void UeiDaq::CUeiDIIndustrialChannel::SetMux (int *line*, tUeiDigitalInputMux *mux*)

Select how voltage supply is connected to each input line. Disconnected: Set the mux to tri-state mode to disconnect voltage supply from input line Diag mode: Set mux to Diagnostic mode to connect internal voltage source to each input line in order to test that it is functional Pull-up mode: Setting mux to pull-up mode connects a pull-up resistor between the internal voltage source and the input line to monitor switches or contacts without external circuitry.

Parameters:

line The input line to configure

mux the new mux state

2.81.2.11 UeiDaqAPI double UeiDaq::CUeiDIIndustrialChannel::GetVoltageSupply (int *line*)

Get the voltage supplied to selected line. Test mode: Guardian feature that connects an internal voltage source to each input line in order to test that it is functional Pull-up mode: connect a pull-up resistor between the internal voltage source and the input line to monitor switch or contacts without external circuitry.

Parameters:

line The input line to query

Returns:

the supplied voltage

2.81.2.12 UeiDaqAPI void UeiDaq::CUeiDIIndustrialChannel::SetVoltageSupply (int *line*, double *voltage*)

Program the voltage supplied to selected line. Test mode: Guardian feature that connects an internal voltage source to each input line in order to test that it is functional Pull-up mode: connect a pull-up resistor between the internal voltage source and the input line to monitor switch or contacts without external circuitry.

Parameters:

line The input line to configure

voltage the voltage to supply to the input line

2.81.2.13 UeiDaqAPI int UeiDaq::CUeiDIIndustrialChannel::GetInputGain (int *line*)

Some DI devices use an ADC to measure the input line state. This setting provides a way to select the input gain for the ADC. The input gain value is an index in the gain list starting with 0

Parameters:

line The input line to query

Returns:

the line's input gain

2.81.2.14 UeiDaqAPI void UeiDaq::CUeiDIIndustrialChannel::SetInputGain (int *line*, int *inputGain*)

Some DI devices use an ADC to measure the input line state. This setting provides a way to select the input gain for the ADC. The input gain value is an index in the gain list starting with 0.

Parameters:

line The input line to configure

inputGain the input gain

2.81.2.15 UeiDaqAPI int UeiDaq::CUeiDIIndustrialChannel::GetMuxDelay (int *line*)

Some DI devices use an ADC to measure the input line state. This setting provides a way to set the delay for the multiplexer in front of the ADC

Parameters:

line The input line to query

Returns:

the line's mux delay in us

2.81.2.16 UeiDaqAPI void UeiDaq::CUeiDIIndustrialChannel::SetMuxDelay (int *line*, int *muxDelayUs*)

Some DI devices use an ADC to measure the input line state. This setting provides a way to set the delay for the multiplexer in front of the ADC

Parameters:

line The input line to configure

muxDelayUs the mux delay in us

The documentation for this class was generated from the following file:

- UeiChannel.h

2.82 UeiDaq::CUeiDMMChannel Class Reference

Manages settings for each DMM input channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiAIChannel**.

Public Member Functions

- UeiDaqAPI **CUeiDMMChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiDMMChannel** ()
  Destructor.
- UeiDaqAPI **tUeiDMMMeasurementMode GetMeasurementMode** ()
  Get the measurement mode.
- UeiDaqAPI void **SetMeasurementMode** (**tUeiDMMMeasurementMode** mode)
  Set the measurement mode.
- UeiDaqAPI **tUeiDMMFIRCutoff GetFIRCutoff** ()
  Get the FIR cutoff frequency.
- UeiDaqAPI void **SetFIRCutoff** (**tUeiDMMFIRCutoff** frequency)
  Set the FIR cutoff frequency.
- UeiDaqAPI **tUeiDMMZeroCrossingMode GetZeroCrossingMode** ()
  Get the FIR decimation factorSet the FIR decimation factorGet the zero crossing detection mode.
- UeiDaqAPI void **SetZeroCrossingMode** (**tUeiDMMZeroCrossingMode** zcMode)
  Set the zero crossing detection mode.
- UeiDaqAPI int **GetZeroCrossingNumberOfSamples** ()
  Get the number of samples used for zero crossing detection.
- UeiDaqAPI void **SetZeroCrossingNumberOfSamples** (int numSamples)
  Set the zero crossing detection mode.
- UeiDaqAPI float **GetZeroCrossingLevel** ()
  Get the zero crossing detection level.
- UeiDaqAPI void **SetZeroCrossingLevel** (float level)
  Set the zero crossing detection mode.
- UeiDaqAPI int **GetPowerLineFrequency** ()
  Get the power line frequency.
- UeiDaqAPI void **SetPowerLineFrequency** (int frequency)

Set the power line frequency.

- UeiDaqAPI int **GetNumberOfPowerLineCycles** ()
  Get the number of power line cycles (NPLC).
- UeiDaqAPI void **SetNumberOfPowerLineCycles** (int numPowerLineCycles)
  Set the number of power line cycles (NPLC).
- UeiDaqAPI int **GetDisconnectTimeout** ()
  Get the number of seconds with no reads before inputs are disconnected.
- UeiDaqAPI void **SetDisconnectTimeout** (int timeoutSeconds)
  Set the number of seconds with no reads before inputs are disconnected.
- UeiDaqAPI bool **IsAutoRangeEnabled** ()
  Get the auto-range state.
- UeiDaqAPI void **EnableAutoRange** (bool enabled)
  Enable or Disable auto-range.

2.82.1 Detailed Description

Manages settings for each DMM input channel

2.82.2 Member Function Documentation

2.82.2.1 UeiDaqAPI tUeiDMMMeasurementMode UeiDaq::CUeiDMMChannel::GetMeasurementMode ()

Get the measurement mode

Returns:

the Measurement mode.

See also:

SetMeasurementMode (p. 232)

2.82.2.2 UeiDaqAPI void UeiDaq::CUeiDMMChannel::SetMeasurementMode (tUeiDMMMeasurementMode *mode*)

Configure the measurement mode.

Parameters:

mode The new measurement mode

See also:

GetMeasurementMode (p. 232)

2.82.2.3 UeiDaqAPI tUeiDMMFIRCutoff UeiDaq::CUeiDMMChannel::GetFIRCutoff ()

Get the FIR cutoff frequency

Returns:

the FIR cutoff frequency

See also:

SetFIRCutoff (p. 233)

2.82.2.4 UeiDaqAPI void UeiDaq::CUeiDMMChannel::SetFIRCutoff (tUeiDMMFIRCutoff *frequency*)

Set the FIR cutoff frequency

Parameters:

frequency The new FIR cutoff frequency

See also:

GetFIRCutoff (p. 233)

2.82.2.5 UeiDaqAPI tUeiDMMZeroCrossingMode UeiDaq::CUeiDMMChannel::GetZeroCrossingMode ()

Zero crossing detection is used for RMS measurements. Get the current mode used to determine the zero crossing detection level.

Returns:

the current ZC detection mode

See also:

SetZeroCrossingMode (p. 233)

2.82.2.6 UeiDaqAPI void UeiDaq::CUeiDMMChannel::SetZeroCrossingMode (tUeiDMMZeroCrossingMode *zcMode*)

Zero crossing detection is used for RMS measurements. Get the current mode used to determine the zero crossing detection level.

Parameters:

zcMode The new ZC detection mode

See also:

GetZeroCrossingMode (p. 233)

2.82.2.7 UeiDaqAPI int UeiDaq::CUeiDMMChannel::GetZeroCrossingNumberOfSamples ()

Zero crossing detection is used for RMS measurements. Get the number of samples used to detect a zero crossing.

Returns:

The number of samples for ZC detection

See also:

SetZeroCrossingNumberOfSamples (p. 234)

2.82.2.8 UeiDaqAPI void UeiDaq::CUeiDMMChannel::SetZeroCrossingNumberOfSamples (int *numSamples*)

Zero crossing detection is used for RMS measurements. Get the current mode used to determine the zero crossing detection level.

Parameters:

numSamples The number of samples for ZC detection

See also:

GetZeroCrossingNumberOfSamples (p. 234)

2.82.2.9 UeiDaqAPI float UeiDaq::CUeiDMMChannel::GetZeroCrossingLevel ()

Zero crossing detection is used for RMS measurements. Get the level used to detect a zero crossing.

Returns:

The ZC level

See also:

SetZeroCrossingLevel (p. 234)

2.82.2.10 UeiDaqAPI void UeiDaq::CUeiDMMChannel::SetZeroCrossingLevel (float *level*)

Zero crossing detection is used for RMS measurements. Get the current mode used to determine the zero crossing detection level.

Parameters:

level The ZC level

See also:

GetZeroCrossingLevel (p. 234)

2.82.2.11 UeiDaqAPI int UeiDaq::CUeiDMMChannel::GetPowerLineFrequency ()

Get the power line frequency used to power the equipment This is used to reject power line induced noise from DC measurements.

Returns:

The power line frequency

See also:

SetPowerLineFrequency (p. 235)

2.82.2.12 UeiDaqAPI void UeiDaq::CUeiDMMChannel::SetPowerLineFrequency (int *frequency*)

Set the power line frequency used to power the equipment This is used to reject power line induced noise from DC measurements.

Parameters:

frequency The power line frequency

See also:

GetPowerLineFrequency (p. 235)

2.82.2.13 UeiDaqAPI int UeiDaq::CUeiDMMChannel::GetNumberOfPowerLineCycles ()

Get the integration time in number of power line cycles This is used to reject power line induced noise from DC measurements.

Returns:

The power line frequency

See also:

SetNumberOfPowerLineCycles (p. 235)

2.82.2.14 UeiDaqAPI void UeiDaq::CUeiDMMChannel::SetNumberOfPowerLineCycles (int *numPowerLineCycles*)

Set the integration time in number of power line cycles This is used to reject power line induced noise from DC measurements.

Parameters:

frequency The power line frequency

See also:

GetNumberOfPowerLineCycles (p. 235)

2.82.2.15 UeiDaqAPI int UeiDaq::CUeiDMMChannel::GetDisconnectTimeout ()

Get the number of seconds with no reads before inputs are disconnected

Returns:

The number of seconds with no reads before inputs are disconnected

See also:

SetDisconnectTimeout (p. 236)

2.82.2.16 UeiDaqAPI void UeiDaq::CUeiDMMChannel::SetDisconnectTimeout (int *timeoutSeconds*)

Set the number of seconds with no reads before inputs are disconnected

Parameters:

timeoutSeconds The number of seconds with no reads before inputs are disconnected

See also:

GetDisconnectTimeout (p. 236)

2.82.2.17 UeiDaqAPI bool UeiDaq::CUeiDMMChannel::IsAutoRangeEnabled ()

Determines whether auto-range is enabled

Returns:

the auto-range state.

See also:

EnableAutoRange (p. 236)

2.82.2.18 UeiDaqAPI void UeiDaq::CUeiDMMChannel::EnableAutoRange (bool *enabled*)

Enable or disable auto-range

Parameters:

enabled The new auto-range state

See also:

IsAutoRangeEnabled (p. 236)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.83 UeiDaq::CUeiDMMReader Class Reference

DMM Reader class.

`#include <UeiReader.h>`

Public Member Functions

- UeiDaqAPI **CUeiDMMReader** (**CUeiDataStream** *pDataStream)
  Constructor.
- virtual UeiDaqAPI ~**CUeiDMMReader** ()
  Destructor.
- UeiDaqAPI void **ReadSingleScan** (**f64** *pBuffer)
  Read a scan.
- UeiDaqAPI void **ReadMultipleScans** (**Int32** numScans, **f64** *pBuffer)
  Read multiple scans.
- UeiDaqAPI void **ReadSingleScanAsync** (**f64** *pBuffer)
  Read a scan asynchronously.
- UeiDaqAPI void **ReadMultipleScansAsync** (**Int32** numScans, **f64** *pBuffer)
  Read multiple scans asynchronously.
- UeiDaqAPI void **ReadStatus** (**uInt32** *pStatus)
  Read DMM status returned with last read data.
- UeiDaqAPI bool **IsProtectionTripped** ()
  Read DMM protection tripped status.
- UeiDaqAPI bool **IsProtectionTrippedMaxRange** ()
  Read DMM protection tripped in max range status.
- UeiDaqAPI bool **IsProtectionReconfiguredSafeRange** ()
  Read DMM protection reconfigured to safe range status.
- UeiDaqAPI void **AddEventListener** (**IUeiEventListener** *pListener)
  Add an asynchronous listener.

2.83.1 Detailed Description

Class that handles reading data of a stream coming from a DMM input

2.83.2 Constructor & Destructor Documentation

2.83.2.1 UeiDaqAPI UeiDaq::CUeiDMMReader::CUeiDMMReader (CUeiDataStream * *pDataStream*)

Parameters:

pDataStream represents the source of data to read

2.83.3 Member Function Documentation

2.83.3.1 UeiDaqAPI void UeiDaq::CUeiDMMReader::ReadSingleScan (f64 * *pBuffer*)

Read only one scan from the input stream

Parameters:

pBuffer destination buffer

2.83.3.2 UeiDaqAPI void UeiDaq::CUeiDMMReader::ReadMultipleScans (Int32 *numScans*, f64 * *pBuffer*)

Read several scans from the input stream

Parameters:

numScans number of scans to read

pBuffer destination buffer

2.83.3.3 UeiDaqAPI void UeiDaq::CUeiDMMReader::ReadSingleScanAsync (f64 * *pBuffer*)

Read only one scan asynchronously from the input stream. The destination buffer contains valid data once the listener is called.

Parameters:

pBuffer destination buffer

2.83.3.4 UeiDaqAPI void UeiDaq::CUeiDMMReader::ReadMultipleScansAsync (Int32 *numScans*, f64 * *pBuffer*)

Read several scans asynchronously from the input stream. The destination buffer contains valid data once the listener is called.

Parameters:

numScans number of scans to read

pBuffer destination buffer

2.83.3.5 UeiDaqAPI void UeiDaq::CUeiDMMReader::ReadStatus (uInt32 * *pStatus*)

Read DMM status returned with last read data. Status is only updated when channel data is read.

Parameters:

pStatus destination for status

2.83.3.6 UeiDaqAPI bool UeiDaq::CUeiDMMReader::IsProtectionTripped ()

Read DMM status returned with last read data and return if protection was tripped when data was read.

Returns:

protection tripped status

2.83.3.7 UeiDaqAPI bool UeiDaq::CUeiDMMReader::IsProtectionTrippedMaxRange ()

Read DMM status returned with last read data and return if protection was tripped in max range when data was read.

Returns:

protection tripped in max range status

2.83.3.8 UeiDaqAPI bool UeiDaq::CUeiDMMReader::IsProtectionReconfiguredSafeRange ()

Read DMM status returned with last read data and return if protection reconfigured to safe range when data was read.

Returns:

protection reconfigured to safe range status

2.83.3.9 UeiDaqAPI void UeiDaq::CUeiDMMReader::AddEventListener (IUeiEventListener * *pListener*)

Subscribe a listener to receive asynchronous events

Parameters:

pListener pointer to a class that implements **IUeiEventListener** (p. 549) interface

The documentation for this class was generated from the following file:

- UeiReader.h

2.84 UeiDaq::CUeiDOChannel Class Reference

Manages settings for each digital output channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Inherited by **UeiDaq::CUeiDOIndustrialChannel**.

Public Member Functions

- UeiDaqAPI **CUeiDOChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiDOChannel** ()
  Destructor.
- UeiDaqAPI bool **IsDefaultValueEnabled** (void)
  Get the default value state.
- UeiDaqAPI void **EnableDefaultValue** (bool enableDefaultValue)
  Set the default value state.
- UeiDaqAPI **uInt32 GetDefaultValue** ()
  Get the default value.
- UeiDaqAPI void **SetDefaultValue** (**uInt32** defaultVal)
  Set the default value.
- UeiDaqAPI **uInt32 GetOutputMask** ()
  Get the line output mask.
- UeiDaqAPI void **SetOutputMask** (**uInt32** mask)
  Set the line output mask.

2.84.1 Detailed Description

Manages settings for each digital output channel

2.84.2 Member Function Documentation

2.84.2.1 UeiDaqAPI bool UeiDaq::CUeiDOChannel::IsDefaultValueEnabled (void)

Determines whether output will be set to a default value when session stops.

Returns:

The default value state.

See also:

EnableDefaultValue (p. 241)

2.84.2.2 UeiDaqAPI void UeiDaq::CUeiDOChannel::EnableDefaultValue (bool *enableDefaultValue*)

Enables default value. The output will be set to a default value when session stops.

Parameters:

enableDefaultValue true to turn stop value on, false otherwise

See also:

IsDefaultValueEnabled (p. 240)

2.84.2.3 UeiDaqAPI uInt32 UeiDaq::CUeiDOChannel::GetDefaultValue ()

Get the default value.

Returns:

the default value.

See also:

SetDefaultValue (p. 241)

2.84.2.4 UeiDaqAPI void UeiDaq::CUeiDOChannel::SetDefaultValue (uInt32 *defaultVal*)

Set the default value.

Parameters:

defaultVal the default value.

See also:

GetDefaultValue (p. 241)

2.84.2.5 UeiDaqAPI uInt32 UeiDaq::CUeiDOChannel::GetOutputMask ()

Some devices allow configuring different direction for lines in the same port. Get the mask used to select wich digital line will be used as output. When bit x is set to 1, line x is used as output.

Returns:

the output mask.

See also:

SetOutputMask (p. 242)

2.84.2.6 UeiDaqAPI void UeiDaq::CUeiDOChannel::SetOutputMask (uInt32 *mask*)

Some devices allow configuring different direction for lines in the same port. Set the mask used to select wich digital line will be used as output. When bit x is set to 1, line x is used as output.

Parameters:

mask the new output mask.

See also:

GetOutputMask (p. 241)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.85 UeiDaq::CUeiDOIndustrialChannel Class Reference

Manages settings for each industrial digital output channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiDOChannel**.

Inherited by **UeiDaq::CUeiDOProtectedChannel**.

Public Member Functions

- UeiDaqAPI **CUeiDOIndustrialChannel** (int numLines)
  Constructor.
- virtual UeiDaqAPI ~**CUeiDOIndustrialChannel** ()
  Destructor.
- UeiDaqAPI **tUeiDOPWMMode GetPWMMode** (int line)
  Get the PWM mode.
- UeiDaqAPI void **SetPWMMode** (int line, **tUeiDOPWMMode** mode)
  Set the PWM mode.
- UeiDaqAPI **uInt32 GetPWMLength** (int line)
  Get the pulse train length.
- UeiDaqAPI void **SetPWMLength** (int line, **uInt32** lengthus)
  Set pulse train length.
- UeiDaqAPI **uInt32 GetSoftStartStopTime** (int line)
  Get the soft start/soft stop pulse train time in micro-seconds.
- UeiDaqAPI void **SetSoftStartStopTime** (int line, **uInt32** timeus)
  Set the soft start/soft stop pulse train time in micro-seconds.
- UeiDaqAPI **uInt32 GetPWMPeriod** (int line)
  Get the pulse train period.
- UeiDaqAPI void **SetPWMPeriod** (int line, **uInt32** periodus)
  Set pulse train period.
- UeiDaqAPI double **GetPWMDutyCycle** (int line)
  Get the pulse train duty cycle.
- UeiDaqAPI void **SetPWMDutyCycle** (int line, double dutyCycle)
  Set pulse train duty cycle.
- UeiDaqAPI **tUeiDigitalTermination GetTermination** (int line)
  Get termination configuration.

- UeiDaqAPI void **SetTermination** (int line, **tUeiDigitalTermination** term)
  Set termination configuration.
- UeiDaqAPI **tUeiDOPWMOutputMode GetPWMOutputMode** (int line)
  Get the PWM ouput mode.
- UeiDaqAPI void **SetPWMOutputMode** (int line, **tUeiDOPWMOutputMode** mode)
  Set the PWM output mode.

2.85.1 Detailed Description

Manages settings for each industrial digital output channel.

2.85.2 Member Function Documentation

2.85.2.1 UeiDaqAPI tUeiDOPWMMode UeiDaq::CUeiDOIndustrialChannel::GetPWMMode (int *line*)

Specifies the PWM mode for DO channels that support the capability With PWM mode you can replace the rising and falling edges with a pulse train to allow for soft start and/or stop

Parameters:

line The output line to configure.

Returns:

The PWM mode.

See also:

SetPWMMode (p. 244)

2.85.2.2 UeiDaqAPI void UeiDaq::CUeiDOIndustrialChannel::SetPWMMode (int *line*, tUeiDOPWMMode *mode*)

Specifies the PWM mode for DO channels that support the capability With PWM mode you can replace the rising and falling edges with a pulse train to allow for soft start and/or stop

Parameters:

line The output line to configure.

mode The new PWM mode.

See also:

GetPWMMode (p. 244)

2.85.2.3 UeiDaqAPI uInt32 UeiDaq::CUeiDOIndustrialChannel::GetPWMLength (int *line*)

Specifies soft start/stop pulse train length in micro-seconds

Parameters:

line The output line to configure.

Returns:

The pulse train length.

See also:

SetPWMLength (p. 245)

2.85.2.4 UeiDaqAPI void UeiDaq::CUeiDOIndustrialChannel::SetPWMLength (int *line*, uInt32 *lengthus*)

Specifies soft start/stop pulse train length in micro-seconds

Parameters:

line The output line to configure.

lengthus The new pulse train length.

See also:

GetPWMLength (p. 245)

2.85.2.5 UeiDaqAPI uInt32 UeiDaq::CUeiDOIndustrialChannel::GetSoftStartStopTime (int *line*)

Specifies soft start/stop pulse train time in micro-seconds Note this is an alias for GetPWMLength

Parameters:

line The output line to configure.

Returns:

The soft start/soft stop pulse train time.

See also:

SetSoftStartStopTime (p. 245)

2.85.2.6 UeiDaqAPI void UeiDaq::CUeiDOIndustrialChannel::SetSoftStartStopTime (int *line*, uInt32 *timeus*)

Specifies soft start/stop pulse train time in micro-seconds Note this is an alias for SetPWMLength

Parameters:

line The output line to configure.

timeus The new soft start/soft stop pulse train time.

See also:

GetSoftStartStopTime (p. 245)

2.85.2.7 UeiDaqAPI uInt32 UeiDaq::CUeiDOIndustrialChannel::GetPWMPeriod (int *line*)

Specifies soft start/stop or continuous pulse train period in micro-seconds

Parameters:

line The output line to configure.

Returns:

The pulse train period.

See also:

SetPWMPeriod (p. 246)

2.85.2.8 UeiDaqAPI void UeiDaq::CUeiDOIndustrialChannel::SetPWMPeriod (int *line*, uInt32 *periodus*)

Specifies soft start/stop or continuous pulse train period in micro-seconds

Parameters:

line The output line to configure.

periodus The new pulse train period.

See also:

GetPWMPeriod (p. 246)

2.85.2.9 UeiDaqAPI double UeiDaq::CUeiDOIndustrialChannel::GetPWMDutyCycle (int *line*)

Specifies the continuous pulse train duty cycle

Parameters:

line The output line to configure.

Returns:

The pulse train period.

See also:

SetPWMDutyCycle (p. 247)

2.85.2.10 UeiDaqAPI void UeiDaq::CUeiDOIndustrialChannel::SetPWMDutyCycle (int *line*, double *dutyCycle*)

Specifies the continuous pulse train duty cycle

Parameters:

line The output line to configure.

dutyCycle The new pulse train duty cycle.

See also:

GetPWMDutyCycle (p. 246)

2.85.2.11 UeiDaqAPI tUeiDigitalTermination UeiDaq::CUeiDOIndustrialChannel::GetTermination (int *line*)

Specifies whether a termination resistor is connected in pull up or down configuration

Parameters:

line The output line to configure.

Returns:

The termination configuration.

See also:

SetTermination (p. 247)

2.85.2.12 UeiDaqAPI void UeiDaq::CUeiDOIndustrialChannel::SetTermination (int *line*, tUeiDigitalTermination *term*)

Specifies whether a termination resistor is connected in pull up or down configuration

Parameters:

line The output line to configure.

term The new termination configuration.

See also:

GetTermination (p. 247)

2.85.2.13 UeiDaqAPI tUeiDOPWMOutputMode UeiDaq::CUeiDOIndustrialChannel::GetPWMOutputMode (int *line*)

Specifies the PWM output mode for DO channels that support the capability

Parameters:

line The output line to configure.

Returns:

The current PWM output mode.

See also:

SetPWMOutputMode (p. 248)

2.85.2.14 UeiDaqAPI void UeiDaq::CUeiDOIndustrialChannel::SetPWMOutputMode (int *line*, tUeiDOPWMOutputMode *mode*)

Specifies the PWM output mode for DO channels that support the capability

Parameters:

line The output line to configure.

mode The new PWM output mode.

See also:

GetPWMOutputMode (p. 247)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.86 UeiDaq::CUeiDOProtectedChannel Class Reference

Manages settings for each protected digital output channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiDOIndustrialChannel**.

Public Member Functions

- UeiDaqAPI **CUeiDOProtectedChannel** (int numLines)
  Constructor.
- virtual UeiDaqAPI ~**CUeiDOProtectedChannel** ()
  Destructor.
- UeiDaqAPI double **GetUnderCurrentLimit** (int line)
  Get the undercurrent limit.
- UeiDaqAPI void **SetUnderCurrentLimit** (int line, double underCurrent)
  Set the undercurrent limit.
- UeiDaqAPI double **GetOverCurrentLimit** (int line)
  Get the overcurrent limit.
- UeiDaqAPI void **SetOverCurrentLimit** (int line, double overCurrent)
  Set the overcurrent limit.
- UeiDaqAPI double **GetCurrentMeasurementRate** (void)
  Get the current measurement rate.
- UeiDaqAPI void **SetCurrentMeasurementRate** (double measurementRate)
  Set the current measurement rate.
- UeiDaqAPI bool **IsCircuitBreakerEnabled** (int line)
  Determines whether the CB is currently protecting the channel.
- UeiDaqAPI void **EnableCircuitBreaker** (int line, bool enable)
  Enable or Disable channel protection.
- UeiDaqAPI bool **GetAutoRetry** (void)
  Get the auto retry setting.
- UeiDaqAPI void **SetAutoRetry** (bool autoRetry)
  Set the auto retry setting.
- UeiDaqAPI double **GetAutoRetryRate** (void)
  Get the auto retry rate.
- UeiDaqAPI void **SetAutoRetryRate** (double autoRetryRate)

Set the auto retry rate.

- UeiDaqAPI **uInt32 GetOverUnderCount** (void)
  Get the over or under limit count.

- UeiDaqAPI void **SetOverUnderCount** (**uInt32** overUnderCount)
  Set the over or under limit count.

2.86.1 Detailed Description

Manages settings for each digital output channel protected by a circuit breaker.

2.86.2 Member Function Documentation

2.86.2.1 UeiDaqAPI double UeiDaq::CUeiDOProtectedChannel::GetUnderCurrentLimit (int *line*)

Get the minimum amount of current allowed in Amps on the specified line of the port associated with the channel object. If less than the minimum current flows through the digital output line, the circuit will open.

Parameters:

line The output line to configure

Returns:

The under current limit.

See also:

SetUnderCurrentLimit (p. 250) **GetOverCurrentLimit** (p. 251) **SetOverCurrentLimit** (p. 251)

2.86.2.2 UeiDaqAPI void UeiDaq::CUeiDOProtectedChannel::SetUnderCurrentLimit (int *line*, double *underCurrent*)

Set the minimum amount of current allowed in Amps on the specified line of the port associated with the channel object. If less than the minimum current flows through the digital output line, the circuit will open.

Parameters:

line The output line to configure

underCurrent The under current limit.

See also:

GetUnderCurrentLimit (p. 250) **GetOverCurrentLimit** (p. 251) **SetOverCurrentLimit** (p. 251)

2.86.2.3 UeiDaqAPI double UeiDaq::CUeiDOProtectedChannel::GetOverCurrentLimit (int *line*)

Get the maximum amount of current allowed in Amps on the specified line of the port associated with the channel object. If more than the maximum current flows through the digital output line, the circuit will open.

Parameters:

line The output line to configure

Returns:

The over current limit.

See also:

SetUnderCurrentLimit (p. 250) **GetUnderCurrentLimit** (p. 250) **SetOverCurrentLimit** (p. 251)

2.86.2.4 UeiDaqAPI void UeiDaq::CUeiDOProtectedChannel::SetOverCurrentLimit (int *line*, double *overCurrent*)

Set the maximum amount of current allowed in Amps on the specified line of the port associated with the channel object. If more than the maximum current flows through the digital output line, the circuit will open.

Parameters:

line The output line to configure

overCurrent The over current limit.

See also:

GetUnderCurrentLimit (p. 250) **GetOverCurrentLimit** (p. 251) **SetUnderCurrentLimit** (p. 250)

2.86.2.5 UeiDaqAPI double UeiDaq::CUeiDOProtectedChannel::GetCurrentMeasurementRate (void)

Get the rate at which the current is measured. This rate determines how fast the device react when an under or over current condition occurs.

Returns:

The current measurement rate (in Hz).

See also:

GetCurrentMeasurementRate (p. 251)

2.86.2.6 UeiDaqAPI void UeiDaq::CUeiDOProtectedChannel::SetCurrentMeasurementRate (double *measurementRate*)

Set the rate at which the current is measured. This rate determines how fast the device react when an under or over current condition occurs.

Parameters:

measurementRate The current measurement rate (in Hz).

See also:

GetCurrentMeasurementRate (p. 251)

2.86.2.7 UeiDaqAPI bool UeiDaq::CUeiDOProtectedChannel::IsCircuitBreakerEnabled (int *line*)

Return true if circuit breaker for this channel is enabled and false otherwise

Parameters:

line The output line to configure

Returns:

Circuit breaker state

2.86.2.8 UeiDaqAPI void UeiDaq::CUeiDOProtectedChannel::EnableCircuitBreaker (int *line*, bool *enable*)

Enable or disable circuit breaker on this channel. when enabled a circuit breaker monitors up a diagnostic channel and opens the circuit if any of the measurements goes out of pre-set limits.

Parameters:

line The output line to configure

enable True to turn-on protection, false to turn it off

2.86.2.9 UeiDaqAPI bool UeiDaq::CUeiDOProtectedChannel::GetAutoRetry (void)

The auto retry setting specifies whether the device should attempt to close the circuit after it was opened because of an over or under current condition. If it is set to true the device will try to close the circuit at a rate set by the autoRetryRate setting.

Returns:

true if autoRetry is on, false otherwise.

See also:

SetAutoRetry (p. 253)

2.86.2.10 UeiDaqAPI void UeiDaq::CUeiDOProtectedChannel::SetAutoRetry (bool *autoRetry*)

The auto retry setting specifies whether the device should attempt to close the circuit after it was opened because of an over or under current condition. If it is set to true the device will try to close the circuit at a rate set by the autoRetryRate setting.

Parameters:

autoRetry true to turn auto-retry on, false to turn it off.

See also:

GetAutoRetry (p. 252)

2.86.2.11 UeiDaqAPI double UeiDaq::CUeiDOProtectedChannel::GetAutoRetryRate (void)

Specifies how often the device will attempt to close a circuit that was opened because of an over or under current condition.

Returns:

The number of retries per second.

See also:

SetAutoRetryRate (p. 253)

2.86.2.12 UeiDaqAPI void UeiDaq::CUeiDOProtectedChannel::SetAutoRetryRate (double *autoRetryRate*)

Specifies how often the device will attempt to close a circuit that was opened because of an over or under current condition.

Parameters:

autoRetryRate The new number of retries per second.

See also:

GetAutoRetryRate (p. 253)

2.86.2.13 UeiDaqAPI uInt32 UeiDaq::CUeiDOProtectedChannel::GetOverUnderCount (void)

Specifies how many consecutive times the monitored current must be out of limits before tripping the circuit breaker.

Returns:

The over or under limit count.

See also:

SetOverUnderCount (p. 254)

2.86.2.14 UeiDaqAPI void UeiDaq::CUeiDOProtectedChannel::SetOverUnderCount (uInt32 *overUnderCount*)

Specifies how many consecutive times the monitored current must be out of limits before tripping the circuit breaker.

Parameters:

overUnderCount The new over or under limit count.

See also:

GetOverUnderCount (p. 253)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.87 UeiDaq::CUeiDriverEnumerator Class Reference

Driver enumerator.

`#include <UeiDriverEnumerator.h>`

Public Member Functions

- UeiDaqAPI **CUeiDriverEnumerator** ()

  Constructor.

- virtual UeiDaqAPI ~**CUeiDriverEnumerator** ()

  Destructor.

- UeiDaqAPI std::string **GetDriver** (int deviceIndex)

  Get a driver resource string from its index.

- UeiDaqAPI int **GetNumberOfDrivers** ()

  Get the number of drivers.

2.87.1 Detailed Description

Enumerates drivers detected by the framework

2.87.2 Constructor & Destructor Documentation

2.87.2.1 UeiDaqAPI UeiDaq::CUeiDriverEnumerator::CUeiDriverEnumerator ()

Build a driver enumerator

2.87.3 Member Function Documentation

2.87.3.1 UeiDaqAPI std::string UeiDaq::CUeiDriverEnumerator::GetDriver (int *deviceIndex*)

Enumerates a driver.

Returns:

a string containing the driver resource or empty if there is no more devices to enumerate.

See also:

GetNumberOfDriverss

2.87.3.2 UeiDaqAPI int UeiDaq::CUeiDriverEnumerator::GetNumberOfDrivers ()

Get the number of enumerated drivers

Returns:

the number of enumerated drivers.

See also:

GetDriver (p. 255)

The documentation for this class was generated from the following file:

- UeiDriverEnumerator.h

2.88 UeiDaq::CUeiException Class Reference

Exception class.

`#include <UeiException.h>`

Public Member Functions

- UeiDaqAPI **CUeiException** (tUeiError error)

  Constructor.

- virtual UeiDaqAPI ~**CUeiException** ()

  Destructor.

- UeiDaqAPI const char * **GetErrorMessage** ()

  Get the error message.

- UeiDaqAPI tUeiError **GetError** ()

  Get the error code.

Static Public Member Functions

- static UeiDaqAPI const char * **TranslateError** (tUeiError error)

  Translate error code.

2.88.1 Detailed Description

This class is used to report exceptions that occur while setting-up or running a session

Examples:

AnalogInBuffered.cpp, **AnalogInBufferedAsync.cpp**, **AnalogInOneShot_Trig_Ex.cpp**, **AnalogInSingle.cpp**, **AnalogOutBuffered.cpp**, **AnalogOutBufferedAsync.cpp**, **AnalogOutSingle.cpp**, **BufferedEventCounting.cpp**, **DigitalInEvent.cpp**, **EventCount.cpp**, **GeneratePulseTrain.cpp**, and **MeasureFrequency.cpp**.

2.88.2 Member Function Documentation

2.88.2.1 UeiDaqAPI const char* UeiDaq::CUeiException::GetErrorMessage ()

Get the error message explaining the reason for the exception

Returns:

error message

Examples:

AnalogInBuffered.cpp, **AnalogInBufferedAsync.cpp**, **AnalogInSingle.cpp**, **AnalogOutBuffered.cpp**, **AnalogOutBufferedAsync.cpp**, **AnalogOutSingle.cpp**, **BufferedEventCounting.cpp**, **DigitalInEvent.cpp**, **EventCount.cpp**, **GeneratePulseTrain.cpp**, and **MeasureFrequency.cpp**.

2.88.2.2 UeiDaqAPI tUeiError UeiDaq::CUeiException::GetError ()

Get the error code of the exception

Returns:

error code

2.88.2.3 static UeiDaqAPI const char* UeiDaq::CUeiException::TranslateError (tUeiError *error*) [static]

Translate the exception error code to a human readable string.

Returns:

error message string

The documentation for this class was generated from the following file:

- UeiException.h

2.89 UeiDaq::CUeiHDLCPort Class Reference

Manages settings for each HDLC port.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Public Member Functions

- UeiDaqAPI **CUeiHDLCPort** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiHDLCPort** ()
  Destructor.
- UeiDaqAPI **tUeiHDLCPortPhysical GetPhysicalInterface** ()
  Get the HDLC port physical interface.
- UeiDaqAPI void **SetPhysicalInterface** (**tUeiHDLCPortPhysical** physInterface)
  Set the HDLC port physical interface.
- UeiDaqAPI **uInt32 GetSpeed** ()
  Get the HDLC port speed.
- UeiDaqAPI void **SetSpeed** (**uInt32** bitsPerSecond)
  Set the HDLC port speed.
- UeiDaqAPI bool **IsTerminationResistorEnabled** ()
  Get the termination resistor state.
- UeiDaqAPI void **EnableTerminationResistor** (bool enabled)
  Enable or disable termination resistor.
- UeiDaqAPI bool **IsHDEchoSuppressionEnabled** ()
  Get the half-duplex echo suppression state.
- UeiDaqAPI void **EnableHDEchoSuppression** (bool enabled)
  Enable or disable half-duplex echo suppression.
- UeiDaqAPI bool **IsLoopbackEnabled** ()
  Get the loopback mode state.
- UeiDaqAPI void **EnableLoopback** (bool enabled)
  Enable or disable half-duplex echo suppression.
- UeiDaqAPI **tUeiHDLCPortAbortSymbol GetAbortSymbol** ()
  Get the HDLC port abort symbol.
- UeiDaqAPI void **SetAbortSymbol** (**tUeiHDLCPortAbortSymbol** abortSymbol)

Set the HDLC port abort symbol.

- UeiDaqAPI **tUeiHDLCPortUnderrunAction GetUnderrunAction** ()
  Get the HDLC port underrun action.
- UeiDaqAPI void **SetUnderrunAction** (**tUeiHDLCPortUnderrunAction** underrunAction)
  Set the HDLC port underrun action.
- UeiDaqAPI **tUeiHDLCPortEncoding GetEncoding** ()
  Get the HDLC port encoding.
- UeiDaqAPI void **SetEncoding** (**tUeiHDLCPortEncoding** encoding)
  Set the HDLC port encoding.
- UeiDaqAPI **tUeiHDLCPortClockSource GetRXClockSource** ()
  Get the HDLC port RX clock source.
- UeiDaqAPI void **SetRXClockSource** (**tUeiHDLCPortClockSource** clockSource)
  Set the HDLC port RX clock source.
- UeiDaqAPI **tUeiHDLCPortClockSource GetTXClockSource** ()
  Get the HDLC port TX clock source.
- UeiDaqAPI void **SetTXClockSource** (**tUeiHDLCPortClockSource** clockSource)
  Set the HDLC port TX clock source.
- UeiDaqAPI **tUeiHDLCPortCRCMode GetCRCMode** ()
  Get the HDLC port CRC mode.
- UeiDaqAPI void **SetCRCMode** (**tUeiHDLCPortCRCMode** crcMode)
  Set the HDLC port CRC mode.
- UeiDaqAPI **tUeiHDLCPortFilterMode GetFilterMode** ()
  Get the HDLC port filter mode.
- UeiDaqAPI void **SetFilterMode** (**tUeiHDLCPortFilterMode** filterMode)
  Set the HDLC port filter mode.
- UeiDaqAPI **uInt8 GetFilterAddress** ()
  Get the HDLC port filter address.
- UeiDaqAPI void **SetFilterAddress** (**uInt8** filterAddress)
  Set the HDLC port filter address.
- UeiDaqAPI **tUeiHDLCPortPreamble GetPreamble** ()
  Get the HDLC port preamble.
- UeiDaqAPI void **SetPreamble** (**tUeiHDLCPortPreamble** preamble)
  Set the HDLC port preamble.

- UeiDaqAPI **tUeiHDLCPortPreambleSize GetPreambleSize** ()

  Get the HDLC port preamble size.

- UeiDaqAPI void **SetPreambleSize** (**tUeiHDLCPortPreambleSize** preambleSize)

  Set the HDLC port preamble size.

- UeiDaqAPI **tUeiHDLCPortIdleCharacter GetIdleCharacter** ()

  Get the HDLC port idle character.

- UeiDaqAPI void **SetIdleCharacter** (**tUeiHDLCPortIdleCharacter** idleChar)

  Set the HDLC port idle character.

2.89.1 Detailed Description

Manages settings for each HDLC port

2.89.2 Member Function Documentation

2.89.2.1 UeiDaqAPI tUeiHDLCPortPhysical UeiDaq::CUeiHDLCPort::GetPhysicalInterface ()

Get the interface used by the HDLC port. Possible interfaces are RS-232, RS-422, RS-485 and V35

Returns:

the HDLC port interface

See also:

SetPhysicalInterface (p. 261)

2.89.2.2 UeiDaqAPI void UeiDaq::CUeiHDLCPort::SetPhysicalInterface (tUeiHDLCPortPhysical *physInterface*)

Set the interface used by the HDLC port. Possible interfaces are RS-232, RS-422, RS-485 and V35

Parameters:

physInterface The new HDLC port interface

See also:

GetPhysicalInterface (p. 261)

2.89.2.3 UeiDaqAPI uInt32 UeiDaq::CUeiHDLCPort::GetSpeed ()

Get the number of data bits transmitted per second.

Returns:

the serial port speed.

See also:

SetSpeed (p. 262)

2.89.2.4 UeiDaqAPI void UeiDaq::CUeiHDLCPort::SetSpeed (uInt32 *bitsPerSecond*)

Set the number of data bits transmitted per second.

Parameters:

bitsPerSecond The serial port speed

See also:

GetSpeed (p. 262)

2.89.2.5 UeiDaqAPI bool UeiDaq::CUeiHDLCPort::IsTerminationResistorEnabled ()

Determines whether the termination resistor is enabled

Returns:

The termination resistor state.

See also:

EnableTerminationResistor (p. 262)

2.89.2.6 UeiDaqAPI void UeiDaq::CUeiHDLCPort::EnableTerminationResistor (bool *enabled*)

Enable or disable the termination resistor.

Parameters:

enabled The new termination resistor state

See also:

IsTerminationResistorEnabled (p. 262)

2.89.2.7 UeiDaqAPI bool UeiDaq::CUeiHDLCPort::IsHDEchoSuppressionEnabled ()

Determines whether the half duplex echo suppression is enabled

Returns:

The half-duplex echo suppression state.

See also:

EnableHDEchoSuppression (p. 263)

2.89.2.8 UeiDaqAPI void UeiDaq::CUeiHDLCPort::EnableHDEchoSuppression (bool *enabled*)

Enable or disable the half-duplex echo suppression.

Parameters:

enabled The new half-duplex echo suppression state

See also:

IsHDEchoSuppressionEnabled (p. 263)

2.89.2.9 UeiDaqAPI bool UeiDaq::CUeiHDLCPort::IsLoopbackEnabled ()

Determines whether the loopback mode is enabled when enabled the port receives what it just transmitted

Returns:

The state.

See also:

EnableLoopback (p. 263)

2.89.2.10 UeiDaqAPI void UeiDaq::CUeiHDLCPort::EnableLoopback (bool *enabled*)

Enable or disable the half-duplex echo suppression. when enabled the port receives what it just transmitted

Parameters:

enabled The new loopback state

See also:

IsLoopbackEnabled (p. 263)

2.89.2.11 UeiDaqAPI tUeiHDLCPortAbortSymbol UeiDaq::CUeiHDLCPort::GetAbortSymbol ()

Get abort symbol used to abort transmission of current frame

Returns:

the current abort symbol

See also:

SetAbortSymbol (p. 264)

2.89.2.12 UeiDaqAPI void UeiDaq::CUeiHDLCPort::SetAbortSymbol (tUeiHDLCPortAbortSymbol *abortSymbol*)

Set the abort symbol used to abort transmission of the current frame

Parameters:

abortSymbol The new abort symbol

See also:

GetAbortSymbol (p. 264)

2.89.2.13 UeiDaqAPI tUeiHDLCPortUnderrunAction UeiDaq::CUeiHDLCPort::GetUnderrunAction ()

Get the action to be taken when an overrun condition occurs

Returns:

the current underrun action

See also:

SetUnderrunAction (p. 264)

2.89.2.14 UeiDaqAPI void UeiDaq::CUeiHDLCPort::SetUnderrunAction (tUeiHDLCPortUnderrunAction *underrunAction*)

Set the action to be taken when an underrun condition occurs

Parameters:

underrunAction The new underrun action

See also:

GetUnderrunAction (p. 264)

2.89.2.15 UeiDaqAPI tUeiHDLCPortEncoding UeiDaq::CUeiHDLCPort::GetEncoding ()

Get the encoding used to transmit data over the serial lines

Returns:

the current encoding

See also:

SetEncoding (p. 265)

2.89.2.16 UeiDaqAPI void UeiDaq::CUeiHDLCPort::SetEncoding (tUeiHDLCPortEncoding *encoding*)

Set the encoding used to transmit data over the serial lines

Parameters:

encoding The new encoding

See also:

GetEncoding (p. 265)

2.89.2.17 UeiDaqAPI tUeiHDLCPortClockSource UeiDaq::CUeiHDLCPort::GetRXClockSource ()

Get the clock source for the receiver

Returns:

the current RX clock source

See also:

SetRXClockSource (p. 265)

2.89.2.18 UeiDaqAPI void UeiDaq::CUeiHDLCPort::SetRXClockSource (tUeiHDLCPortClockSource *clockSource*)

Set the clock source for the receiver

Parameters:

clockSource The new RX clock source

See also:

GetRXClockSource (p. 265)

2.89.2.19 UeiDaqAPI tUeiHDLCPortClockSource UeiDaq::CUeiHDLCPort::GetTXClockSource ()

Get the clock source for the transmitter

Returns:

the current TX clock source

See also:

SetTXClockSource (p. 266)

2.89.2.20 UeiDaqAPI void UeiDaq::CUeiHDLCPort::SetTXClockSource (tUeiHDLCPortClockSource *clockSource*)

Set the clock source for the transmitter

Parameters:

clockSource The new TX clock source

See also:

GetRXClockSource (p. 265)

2.89.2.21 UeiDaqAPI tUeiHDLCPortCRCMode UeiDaq::CUeiHDLCPort::GetCRCMode ()

Get the method used to calculate HDLC frame's CRC

Returns:

the current CRC mode

See also:

SetCRCMode (p. 266)

2.89.2.22 UeiDaqAPI void UeiDaq::CUeiHDLCPort::SetCRCMode (tUeiHDLCPortCRCMode *crcMode*)

Set the method used to calculate HDLC frame's CRC

Parameters:

crcMode The new CRC mode

See also:

GetCRCMode (p. 266)

2.89.2.23 UeiDaqAPI tUeiHDLCPortFilterMode UeiDaq::CUeiHDLCPort::GetFilterMode ()

Get the mode used to filter incoming frames

Returns:

the current filter mode

See also:

SetFilterMode (p. 267)

2.89.2.24 UeiDaqAPI void UeiDaq::CUeiHDLCPort::SetFilterMode (tUeiHDLCPortFilterMode *filterMode*)

Set the mode used to filter incoming frames

Parameters:

filterMode The new filter mode

See also:

GetFilterMode (p. 267)

2.89.2.25 UeiDaqAPI uInt8 UeiDaq::CUeiHDLCPort::GetFilterAddress ()

Get the filter address

Returns:

the current filter address

See also:

SetFilterAddress (p. 267)

2.89.2.26 UeiDaqAPI void UeiDaq::CUeiHDLCPort::SetFilterAddress (uInt8 *filterAddress*)

Set the filter address

Parameters:

filterAddress The new filter address

See also:

GetFilterAddress (p. 267)

2.89.2.27 UeiDaqAPI tUeiHDLCPortPreamble UeiDaq::CUeiHDLCPort::GetPreamble ()

Get the preamble

Returns:

the current preamble

See also:

SetPreamble (p. 268)

2.89.2.28 UeiDaqAPI void UeiDaq::CUeiHDLCPort::SetPreamble (tUeiHDLCPortPreamble *preamble*)

Set the preamble

Parameters:

preamble The new preamble

See also:

GetPreamble (p. 268)

2.89.2.29 UeiDaqAPI tUeiHDLCPortPreambleSize UeiDaq::CUeiHDLCPort::GetPreambleSize ()

Get the preamble size

Returns:

the current preamble size

See also:

SetPreambleSize (p. 268)

2.89.2.30 UeiDaqAPI void UeiDaq::CUeiHDLCPort::SetPreambleSize (tUeiHDLCPortPreambleSize *preambleSize*)

Set the preamble size

Parameters:

preambleSize The new preamble size

See also:

GetPreambleSize (p. 268)

2.89.2.31 UeiDaqAPI tUeiHDLCPortIdleCharacter UeiDaq::CUeiHDLCPort::GetIdleCharacter ()

Get the idle character

Returns:

the current idle character

See also:

SetIdleCharacter (p. 269)

2.89.2.32 UeiDaqAPI void UeiDaq::CUeiHDLCPort::SetIdleCharacter (tUeiHDLCPortIdleCharacter *idleChar*)

Set the idle character

Parameters:

idleChar The new idle character

See also:

GetIdleCharacter (p. 269)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.90 UeiDaq::CUeiHDLCReader Class Reference

HDLC Reader class.

`#include <UeiMessaging.h>`

Public Member Functions

- UeiDaqAPI **CUeiHDLCReader** (**CUeiDataStream** *pDataStream, **Int32** port=0)
  Constructor.
- virtual UeiDaqAPI ~**CUeiHDLCReader** ()
  Destructor.
- UeiDaqAPI void **Read** (**Int32** bufferSize, **Int8** *pBuffer, **Int32** *numBytesRead)
  Read data from the HDLC port.
- UeiDaqAPI void **ReadAsync** (**Int32** bufferSize, **Int8** *pBuffer)
  Read data from the HDLC port asynchronously.
- UeiDaqAPI void **AddEventListener** (**IUeiEventListener** *pListener)
  Add an asynchronous listener.

2.90.1 Detailed Description

Class that handles reading data messages of a stream coming from a HDLC interface

2.90.2 Constructor & Destructor Documentation

2.90.2.1 UeiDaqAPI UeiDaq::CUeiHDLCReader::CUeiHDLCReader (CUeiDataStream * *pDataStream*, Int32 *port* = 0)

Parameters:

pDataStream represents the source of data to read
port The HDLC port to read data from

2.90.3 Member Function Documentation

2.90.3.1 UeiDaqAPI void UeiDaq::CUeiHDLCReader::Read (Int32 *bufferSize*, Int8 * *pBuffer*, Int32 * *numBytesRead*)

Read a data message from the input stream

Parameters:

bufferSize the size of the destination buffer
pBuffer destination buffer
numBytesRead the actual number of bytes read from the HDLC port

2.90.3.2 UeiDaqAPI void UeiDaq::CUeiHDLCReader::ReadAsync (Int32 *bufferSize*, Int8 * *pBuffer*)

Read a data message from the input stream

Parameters:

bufferSize the size of the destination buffer

pBuffer destination buffer

2.90.3.3 UeiDaqAPI void UeiDaq::CUeiHDLCReader::AddEventListener (IUeiEventListener * *pListener*)

Subscribe a listener to receive asynchronous events

Parameters:

pListener pointer to a class that implements **IUeiEventListener** (p. 549) interface

The documentation for this class was generated from the following file:

- UeiMessaging.h

2.91 UeiDaq::CUeiHDLCWriter Class Reference

HDLC Writer class.

`#include <UeiMessaging.h>`

Public Member Functions

- UeiDaqAPI **CUeiHDLCWriter** (**CUeiDataStream** *pDataStream, **Int32** port=0)
  Constructor.
- virtual UeiDaqAPI ~**CUeiHDLCWriter** ()
  Destructor.
- UeiDaqAPI void **Write** (**Int32** bufferSize, **Int8** *pBuffer, **Int32** *numBytesWritten)
  Write data to the HDLC port.
- UeiDaqAPI void **WriteAsync** (**Int32** bufferSize, **Int8** *pBuffer)
  Write data to the HDLC port asynchronously.
- UeiDaqAPI void **AddEventListener** (**IUeiEventListener** *pListener)
  Add an asynchronous listener.

2.91.1 Detailed Description

Class that handles writing data messages to a stream going through a HDLC interface

2.91.2 Constructor & Destructor Documentation

2.91.2.1 UeiDaqAPI UeiDaq::CUeiHDLCWriter::CUeiHDLCWriter (CUeiDataStream * *pDataStream*, Int32 *port* = 0)

Parameters:

pDataStream represents the destination where to write data
port the HDLC port to write data to

2.91.3 Member Function Documentation

2.91.3.1 UeiDaqAPI void UeiDaq::CUeiHDLCWriter::Write (Int32 *bufferSize*, Int8 * *pBuffer*, Int32 * *numBytesWritten*)

Write a data message to the HDLC poirt

Parameters:

bufferSize the size of the source buffer
pBuffer source buffer
numBytesWritten the actual number of bytes written to the HDLC port

2.91.3.2 UeiDaqAPI void UeiDaq::CUeiHDLCWriter::WriteAsync (Int32 *bufferSize*, Int8 * *pBuffer*)

Write a data message to the HDLC port

Parameters:

bufferSize the size of the source buffer

pBuffer source buffer

2.91.3.3 UeiDaqAPI void UeiDaq::CUeiHDLCWriter::AddEventListener (IUeiEventListener * *pListener*)

Subscribe a listener to receive asynchronous events

Parameters:

pListener pointer to a class that implements **IUeiEventListener** (p. 549) interface

The documentation for this class was generated from the following file:

- UeiMessaging.h

2.92 UeiDaq::CUeiI2CMasterPort Class Reference

Manage I2C master port settings.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Public Member Functions

- UeiDaqAPI **CUeiI2CMasterPort** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiI2CMasterPort** ()
  Destructor.
- UeiDaqAPI **tUeiI2CPortSpeed GetSpeed** ()
  Get the I2C bus bit rate.
- UeiDaqAPI void **SetSpeed** (**tUeiI2CPortSpeed** bitRate)
  Set the I2C bus bit rate.
- UeiDaqAPI float **GetCustomSpeed** ()
  Get the I2C bus custom bit rate.
- UeiDaqAPI void **SetCustomSpeed** (float bitRate)
  Set the I2C bus custom bit rate.
- UeiDaqAPI **tUeiI2CTTLLevel GetTTLLevel** ()
  Get the I2C bus TTL level.
- UeiDaqAPI void **SetTTLLevel** (**tUeiI2CTTLLevel** ttlLevel)
  Set the I2C bus TTL level.
- UeiDaqAPI bool **IsSecureShellEnabled** ()
  Query the secure shell feature.
- UeiDaqAPI void **EnableSecureShell** (bool enabled)
  Enable or Disable secure shell feature.
- UeiDaqAPI bool **IsTerminationResistorEnabled** ()
  Query the termination resistor feature.
- UeiDaqAPI void **EnableTerminationResistor** (bool enabled)
  Enable or Disable termination resistor.
- UeiDaqAPI **tUeiI2CLoopback GetLoopbackMode** ()
  Get the loopback mode.
- UeiDaqAPI void **SetLoopbackMode** (**tUeiI2CLoopback** mode)

Get the loopback mode.

- UeiDaqAPI **uInt16 GetByteToByteDelay** ()

  Get the delay in microseconds that the master will delay between bytes.

- UeiDaqAPI void **SetByteToByteDelay** (**uInt16** delayUs)

  Set the delay in microseconds that the master will delay between bytes.

- UeiDaqAPI **uInt16 GetMaxClockStretchingDelay** ()

  Get the time in microseconds that the master will allow a slave to delay the clock.

- UeiDaqAPI void **SetMaxClockStretchingDelay** (**uInt16** delayUs)

  Set the time in microseconds that the master will allow a slave to delay the clock.

2.92.1 Detailed Description

This objects stores I2C master port settings

2.92.2 Member Function Documentation

2.92.2.1 UeiDaqAPI tUeiI2CPortSpeed UeiDaq::CUeiI2CMasterPort::GetSpeed ()

Get the number of data bits transmitted per second.

Returns:

the I2C bus bit rate.

2.92.2.2 UeiDaqAPI void UeiDaq::CUeiI2CMasterPort::SetSpeed (tUeiI2CPortSpeed *bitRate*)

Set the number of data bits transmitted per second.

Parameters:

bitRate the new I2C bus bit rate.

2.92.2.3 UeiDaqAPI float UeiDaq::CUeiI2CMasterPort::GetCustomSpeed ()

Get the number of data bits transmitted per second.

Returns:

the I2C bus custom bit rate.

2.92.2.4 UeiDaqAPI void UeiDaq::CUeiI2CMasterPort::SetCustomSpeed (float *bitRate*)

Set the number of data bits transmitted per second.

Parameters:

bitRate the new I2C bus custom bit rate.

2.92.2.5 UeiDaqAPI tUeiI2CTTLLevel UeiDaq::CUeiI2CMasterPort::GetTTLLevel ()

Get the TTL level for data and clock signals.

Returns:

the TTL level.

2.92.2.6 UeiDaqAPI void UeiDaq::CUeiI2CMasterPort::SetTTLLevel (tUeiI2CTTLLevel *ttlLevel*)

Set the TTL level for data and clock signals.

Parameters:

ttlLevel the new TTL level.

2.92.2.7 UeiDaqAPI bool UeiDaq::CUeiI2CMasterPort::IsSecureShellEnabled ()

Query status of secure shell feature.

Returns:

the secure shell status.

2.92.2.8 UeiDaqAPI void UeiDaq::CUeiI2CMasterPort::EnableSecureShell (bool *enabled*)

Enable or Disable secure shell.

Parameters:

enabled the new secure shell state.

2.92.2.9 UeiDaqAPI bool UeiDaq::CUeiI2CMasterPort::IsTerminationResistorEnabled ()

Query the termination resistor feature.

Returns:

the current termination resistor status.

2.92.2.10 UeiDaqAPI void UeiDaq::CUeiI2CMasterPort::EnableTerminationResistor (bool *enabled*)

Enable or Disable termination resistor.

Parameters:

enabled the new termination resistor state.

2.92.2.11 UeiDaqAPI tUeiI2CLoopback UeiDaq::CUeiI2CMasterPort::GetLoopbackMode ()

Get the loopback mode.

Returns:

the current loopback mode.

2.92.2.12 UeiDaqAPI void UeiDaq::CUeiI2CMasterPort::SetLoopbackMode (tUeiI2CLoopback *mode*)

Get the loopback mode.

Parameters:

mode the new loopback mode.

2.92.2.13 UeiDaqAPI uInt16 UeiDaq::CUeiI2CMasterPort::GetByteToByteDelay ()

Get the delay in microseconds that the master will delay between sending bytes

Returns:

the current delay in microseconds

2.92.2.14 UeiDaqAPI void UeiDaq::CUeiI2CMasterPort::SetByteToByteDelay (uInt16 *delayUs*)

Set the delay in microseconds that the master will delay between sending bytes.

Parameters:

delayUs the new delay in microseconds

2.92.2.15 UeiDaqAPI uInt16 UeiDaq::CUeiI2CMasterPort::GetMaxClockStretchingDelay ()

Get the time in microseconds that the master will allow a slave to delay the clock

Returns:

the current time in microseconds

2.92.2.16 UeiDaqAPI void UeiDaq::CUeiI2CMasterPort::SetMaxClockStretchingDelay (uInt16 *delayUs*)

Set the time in microseconds that the master will allow a slave to delay the clock

Parameters:

delayUs the new time in microseconds

The documentation for this class was generated from the following file:

- UeiChannel.h

2.93 UeiDaq::CUeiI2CSlavePort Class Reference

Manage I2C slave port settings.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Public Member Functions

- UeiDaqAPI **CUeiI2CSlavePort** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiI2CSlavePort** ()
  Destructor.
- UeiDaqAPI **tUeiI2CTTLLevel GetTTLLevel** ()
  Get the I2C bus TTL level.
- UeiDaqAPI void **SetTTLLevel** (**tUeiI2CTTLLevel** ttlLevel)
  Set the I2C bus TTL level.
- UeiDaqAPI **tUeiI2CSlaveAddressWidth GetSlaveAddressWidth** ()
  Get the slave address width.
- UeiDaqAPI void **SetSlaveAddressWidth** (**tUeiI2CSlaveAddressWidth** width)
  Set the I2C slave address width.
- UeiDaqAPI int **GetSlaveAddress** ()
  Get the slave address.
- UeiDaqAPI void **SetSlaveAddress** (int address)
  Set the slave address.
- UeiDaqAPI bool **IsBusMonitorEnabled** ()
  Query the bus monitor feature.
- UeiDaqAPI void **EnableBusMonitor** (bool enabled)
  Enable or disable bus monitor.
- UeiDaqAPI bool **IsBusMonitorAckEnabled** ()
  Query the bus monitor ACK feature.
- UeiDaqAPI void **EnableBusMonitorAck** (bool enabled)
  Enable or disable bus monitor ACK.
- UeiDaqAPI **tUeiI2CSlaveDataMode GetTransmitDataMode** ()
  Get slave transmit data mode.
- UeiDaqAPI void **SetTransmitDataMode** (**tUeiI2CSlaveDataMode** dataMode)

Set slave transmit data mode.

- UeiDaqAPI **uInt8 GetSlaveRegisterData** (int byteIndex)
  Get data transmitted when in Register mode or padding data when in FIFO mode.
- UeiDaqAPI void **SetSlaveRegisterData** (int byteIndex, **uInt8** data)
  Set data transmitted when in Register mode or padding data when in FIFO mode.
- UeiDaqAPI int **GetSlaveRegisterDataSize** ()
  Get the number of bytes used in slave data when in Register data mode.
- UeiDaqAPI void **SetSlaveRegisterDataSize** (int size)
  Set the number of bytes used in slave data when in Register data mode.
- UeiDaqAPI int **GetMaxWordsPerAck** ()
  Get maximum number of words slave will receive per transmission.
- UeiDaqAPI void **SetMaxWordsPerAck** (int maxWords)
  Set maximum number of words slave will receive per transmission.
- UeiDaqAPI **uInt16 GetClockStretchingDelay** ()
  Get the number of clocks the slave will delay an ACK.
- UeiDaqAPI void **SetClockStretchingDelay** (**uInt16** clocks)
  Get the number of clocks the slave will delay an ACK.
- UeiDaqAPI bool **IsAddressClockStretchingEnabled** ()
  Query the status of clock stretching during address cycle.
- UeiDaqAPI void **EnableAddressClockStretching** (bool enabled)
  Enable clock stretching during address cycle.
- UeiDaqAPI bool **IsTransmitClockStretchingEnabled** ()
  Query the status of clock stretching during transmit cycle.
- UeiDaqAPI void **EnableTransmitClockStretching** (bool enabled)
  Enable clock stretching during transmit cycle.
- UeiDaqAPI bool **IsReceiveClockStretchingEnabled** ()
  Query the status of clock stretching during receive cycle.
- UeiDaqAPI void **EnableReceiveClockStretching** (bool enabled)
  Enable clock stretching during receive cycle.

2.93.1 Detailed Description

This objects stores I2C slave port settings

2.93.2 Member Function Documentation

2.93.2.1 UeiDaqAPI tUeiI2CTTLLevel UeiDaq::CUeiI2CSlavePort::GetTTLLevel ()

Get the TTL level for data and clock signals.

Returns:

the TTL level.

2.93.2.2 UeiDaqAPI void UeiDaq::CUeiI2CSlavePort::SetTTLLevel (tUeiI2CTTLLevel *ttlLevel*)

Set the TTL level for data and clock signals.

Parameters:

ttlLevel the new TTL level.

2.93.2.3 UeiDaqAPI tUeiI2CSlaveAddressWidth UeiDaq::CUeiI2CSlavePort::GetSlaveAddressWidth ()

Get the slave address width.

Returns:

the slave address width.

2.93.2.4 UeiDaqAPI void UeiDaq::CUeiI2CSlavePort::SetSlaveAddressWidth (tUeiI2CSlaveAddressWidth *width*)

Set the slave address width.

Parameters:

width the new slave address width.

2.93.2.5 UeiDaqAPI int UeiDaq::CUeiI2CSlavePort::GetSlaveAddress ()

Get the slave address.

Returns:

the slave address.

2.93.2.6 UeiDaqAPI void UeiDaq::CUeiI2CSlavePort::SetSlaveAddress (int *address*)

Set the slave address.

Parameters:

address the new slave address.

2.93.2.7 UeiDaqAPI bool UeiDaq::CUeiI2CSlavePort::IsBusMonitorEnabled ()

Query the status of the bus monitor feature.

Returns:

the status of the bus monitor feature.

2.93.2.8 UeiDaqAPI void UeiDaq::CUeiI2CSlavePort::EnableBusMonitor (bool *enabled*)

Enable or disable bus monitor.

Parameters:

enabled the new bus monitor state.

2.93.2.9 UeiDaqAPI bool UeiDaq::CUeiI2CSlavePort::IsBusMonitorAckEnabled ()

Query the status of the bus monitor ACK feature. If enabled, bus monitor will generate an ACK on receiving data or address to allow simulating multiple devices on the bus.

Returns:

the status of the bus monitor ACK feature.

2.93.2.10 UeiDaqAPI void UeiDaq::CUeiI2CSlavePort::EnableBusMonitorAck (bool *enabled*)

Enable or disable bus monitor ACK. If enabled, bus monitor will generate an ACK on receiving data or address to allow simulating multiple devices on the bus.

Parameters:

enabled the new bus monitor ACK state.

2.93.2.11 UeiDaqAPI tUeiI2CSlaveDataMode UeiDaq::CUeiI2CSlavePort::GetTransmitDataMode ()

Get slave transmit data mode

Returns:

the slave transmit data mode

2.93.2.12 UeiDaqAPI void UeiDaq::CUeiI2CSlavePort::SetTransmitDataMode (tUeiI2CSlaveDataMode *dataMode*)

Set slave transmit data mode

Parameters:

dataMode the new slave transmit data mode

2.93.2.13 UeiDaqAPI uInt8 UeiDaq::CUeiI2CSlavePort::GetSlaveRegisterData (int *byteIndex*)

Get data transmitted when in Register mode or padding data when in FIFO mode.

Parameters:

byteIndex index of the byte to retrieve, 0-3

Returns:

data used in Register mode of padding data in FIFO mode.

2.93.2.14 UeiDaqAPI void UeiDaq::CUeiI2CSlavePort::SetSlaveRegisterData (int *byteIndex*, uInt8 *data*)

Set data transmitted when in Register mode or padding data when in FIFO mode.

Parameters:

byteIndex index of the byte to set, 0-3

data the new byte data to use

2.93.2.15 UeiDaqAPI int UeiDaq::CUeiI2CSlavePort::GetSlaveRegisterDataSize ()

Get the number of bytes used in slave data when in Register data mode.

Returns:

the number of bytes used in slave data when in Register data mode

2.93.2.16 UeiDaqAPI void UeiDaq::CUeiI2CSlavePort::SetSlaveRegisterDataSize (int *size*)

Set the number of bytes used in slave data when in Register data mode.

Parameters:

size the new number of bytes used in slave data when in Register data mode

2.93.2.17 UeiDaqAPI int UeiDaq::CUeiI2CSlavePort::GetMaxWordsPerAck ()

Get maximum number of words slave will receive per transmission

Returns:

the maximum number of words slave will receive per transmission

2.93.2.18 UeiDaqAPI void UeiDaq::CUeiI2CSlavePort::SetMaxWordsPerAck (int *maxWords*)

Set maximum number of words slave will receive per transmission

Parameters:

maxWords the new maximum number of words slave will receive per transmission

2.93.2.19 UeiDaqAPI uInt16 UeiDaq::CUeiI2CSlavePort::GetClockStretchingDelay ()

Get the delay in 15ns clock increments that the slave will delay an ACK. The clock stretching delay must also be individually enabled for address, transmit, and receive cycles. Max clock stretching delay is ~62ms.

Returns:

the current delay in clocks

2.93.2.20 UeiDaqAPI void UeiDaq::CUeiI2CSlavePort::SetClockStretchingDelay (uInt16 *clocks*)

Get the delay in 15ns clock increments that the slave will delay an ACK. The clock stretching delay must also be individually enabled for address, transmit, and receive cycles. Max clock stretching delay is ~62ms.

Parameters:

clocks the new delay in clocks

2.93.2.21 UeiDaqAPI bool UeiDaq::CUeiI2CSlavePort::IsAddressClockStretchingEnabled ()

Query the status of clock stretching during address cycle.

Returns:

the status of clock stretching during address cycle.

2.93.2.22 UeiDaqAPI void UeiDaq::CUeiI2CSlavePort::EnableAddressClockStretching (bool *enabled*)

Enable clock stretching during address cycle.

Parameters:

enabled the new clock stretching during address cycle state

2.93.2.23 UeiDaqAPI bool UeiDaq::CUeiI2CSlavePort::IsTransmitClockStretchingEnabled ()

Query the status of clock stretching during transmit cycle.

Returns:

the status of clock stretching during transmit cycle.

2.93.2.24 UeiDaqAPI void UeiDaq::CUeiI2CSlavePort::EnableTransmitClockStretching (bool *enabled*)

Enable clock stretching during transmit cycle.

Parameters:

enabled the new clock stretching during transmit cycle state

2.93.2.25 UeiDaqAPI bool UeiDaq::CUeiI2CSlavePort::IsReceiveClockStretchingEnabled ()

Query the status of clock stretching during receive cycle.

Returns:

the status of clock stretching during receive cycle.

2.93.2.26 UeiDaqAPI void UeiDaq::CUeiI2CSlavePort::EnableReceiveClockStretching (bool *enabled*)

Enable clock stretching during receive cycle.

Parameters:

enabled the new clock stretching during receive cycle state

The documentation for this class was generated from the following file:

- UeiChannel.h

2.94 UeiDaq::CUeiIRIGDOTTLChannel Class Reference

Manages settings for each IRIG digital TTL ouput port.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Public Member Functions

- UeiDaqAPI **CUeiIRIGDOTTLChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiIRIGDOTTLChannel** ()
  Destructor.
- UeiDaqAPI **tUeiIRIGDOTTLSource GetSource** (int line)
  Get the source for the specified TTL output.
- UeiDaqAPI void **SetSource** (int line, **tUeiIRIGDOTTLSource** source)
  Set the source for the specified TTL output.
- UeiDaqAPI bool **IsTwoTTLBuffersEnabled** (void)
  Get the TTL buffers state.
- UeiDaqAPI void **EnableTwoTTLBuffers** (bool enableTwoTTLBuffers)
  Set the TTL buffers state.
- UeiDaqAPI bool **Is40nsPulseEnabled** (int line)
  Get the pulse shape state.
- UeiDaqAPI void **Enable40nsPulse** (int line, bool enable40nsPulse)
  Set the pulse shape state.
- UeiDaqAPI bool **IsSyncLineDriven** (int line)
  Get the sync line state for the specified TTL output.
- UeiDaqAPI void **DriveSyncLine** (int line, bool driveSyncLine)
  Set the sync line state for the specified TTL output.
- UeiDaqAPI int **GetEventModuleRate** (int eventChannel)
  Get event module PLL output rate.
- UeiDaqAPI void **SetEventModuleRate** (int eventChannel, int rate)
  Set event module PLL output rate.
- UeiDaqAPI **tUeiIRIGEventSource GetEventModuleSource** (int eventChannel)
  Get event module reference clock.
- UeiDaqAPI void **SetEventModuleSource** (int eventChannel, **tUeiIRIGEventSource** source)

Set event module reference clock.

2.94.1 Detailed Description

Manages settings for each IRIG digital TTL ouput port

2.94.2 Member Function Documentation

2.94.2.1 UeiDaqAPI tUeiIRIGDOTTLSource UeiDaq::CUeiIRIGDOTTLChannel::GetSource (int *line*)

Get the source for the specified TTL output

Parameters:

line the TTL output line

Returns:

the TTL source.

See also:

SetSource (p. 287)

2.94.2.2 UeiDaqAPI void UeiDaq::CUeiIRIGDOTTLChannel::SetSource (int *line*, tUeiIRIGDOTTLSource *source*)

Set the source for the specified TTL output

Parameters:

line the TTL output line

source the new source.

See also:

GetSource (p. 287)

2.94.2.3 UeiDaqAPI bool UeiDaq::CUeiIRIGDOTTLChannel::IsTwoTTLBuffersEnabled (void)

Determines whether one or two TTL buffers are used to drive TTL lines. Enabling two drivers together provide stronger driving capabilities

Returns:

The TTL buffers state.

See also:

EnableTwoTTLBuffers (p. 288)

2.94.2.4 UeiDaqAPI void UeiDaq::CUeiIRIGDOTTLChannel::EnableTwoTTLBuffers (bool *enableTwoTTLBuffers*)

If selected, two TTL buffers are used to drive TTL lines. Enabling two drivers together provide stronger driving capabilities

Parameters:

enableTwoTTLBuffers true to use two buffers, false to use one buffer

See also:

IsTwoTTLBuffersEnabled (p. 287)

2.94.2.5 UeiDaqAPI bool UeiDaq::CUeiIRIGDOTTLChannel::Is40nsPulseEnabled (int *line*)

Determines whether TTL pulses are 60us or 40ns wide

Parameters:

line the TTL output line

Returns:

The TTL buffers state.

See also:

Enable40nsPulse (p. 288)

2.94.2.6 UeiDaqAPI void UeiDaq::CUeiIRIGDOTTLChannel::Enable40nsPulse (int *line*, bool *enable40nsPulse*)

If selected, TTL pulses are 40ns wide otherwise they are 60us wide by default

Parameters:

line the TTL output line

enable40nsPulse true to emit 40ns pulses, false to emit 60us pulses

See also:

Is40nsPulseEnabled (p. 288)

2.94.2.7 UeiDaqAPI bool UeiDaq::CUeiIRIGDOTTLChannel::IsSyncLineDriven (int *line*)

Determines whether the specified TTL output will be routed to the corresponding SYNc line

Parameters:

line the TTL output line

Returns:

The sync line state.

See also:

DriveSyncLine (p. 289)

2.94.2.8 UeiDaqAPI void UeiDaq::CUeiIRIGDOTTLChannel::DriveSyncLine (int *line*, bool *driveSyncLine*)

If selected, the specified TTL output is routed to the corresponding SYNc line

Parameters:

line the TTL output line

driveSyncLine true to emit 40ns pulses, false to emit 60us pulses

See also:

IsSyncLineDriven (p. 288)

2.94.2.9 UeiDaqAPI int UeiDaq::CUeiIRIGDOTTLChannel::GetEventModuleRate (int *eventChannel*)

The event module creates timed interrupts and provides synchronization signals to the outside world through the TTL outputs or the sync lines. The event module includes four event channels, each channel is equipped with a DPLL that can generate a clock signal synchronized with a much slower reference clock such as the 1PPS signal fron the IRIG input or from the GPS input The rate is pecified in number of output pulses per reference pulse.

Parameters:

eventChannel the event channel to configure

Returns:

the current rate of the signal generated by the event module PLL

2.94.2.10 UeiDaqAPI void UeiDaq::CUeiIRIGDOTTLChannel::SetEventModuleRate (int *eventChannel*, int *rate*)

Parameters:

eventChannel the event channel to configure

rate the rate of the signal generated by the event module PLL

2.94.2.11 UeiDaqAPI tUeiIRIGEventSource UeiDaq::CUeiIRIGDOTTLChannel::GetEventModuleSource (int *eventChannel*)

The event module creates timed interrupts and provides synchronization signals to the outside world through the TTL outputs or the sync lines. The event module includes four event channels, each channel is equipped with a DPLL that can generate a clock signal synchronized with a much slower reference clock such as the 1PPS signal fron the IRIG input or from the GPS input

Parameters:

eventChannel the event channel to configure

Returns:

the current source of the event module PLL reference clock

2.94.2.12 UeiDaqAPI void UeiDaq::CUeiIRIGDOTTLChannel::SetEventModuleSource (int *eventChannel*, tUeiIRIGEventSource *source*)

The event module creates timed interrupts and provides synchronization signals to the outside world through the TTL outputs or the sync lines. The event module includes four event channels, each channel is equipped with a DPLL that can generate a clock signal synchronized with a much slower reference clock such as the 1PPS signal fron the IRIG input or from the GPS input

Parameters:

eventChannel the event channel to configure

source the source of the event module PLL reference clock

The documentation for this class was generated from the following file:

- UeiChannel.h

2.95 UeiDaq::CUeiIRIGInputChannel Class Reference

Manages settings for each IRIG time code input channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Public Member Functions

- UeiDaqAPI **CUeiIRIGInputChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiIRIGInputChannel** ()
  Destructor.
- UeiDaqAPI **tUeiIRIGDecoderInputType GetTimeDecoderInput** ()
  Get the time decoder input type.
- UeiDaqAPI void **SetTimeDecoderInput** (**tUeiIRIGDecoderInputType** input)
  Set the time decoder input type.
- UeiDaqAPI **tUeiIRIGTimeCodeFormat GetTimeCodeFormat** ()
  Get the time code format.
- UeiDaqAPI void **SetTimeCodeFormat** (**tUeiIRIGTimeCodeFormat** format)
  Set the time decoder input type.
- UeiDaqAPI bool **IsExternalClockEnabled** (void)
  Get the external clock state.
- UeiDaqAPI void **EnableExternalClock** (bool enableExternalClock)
  Set the external clock state.
- UeiDaqAPI bool **IsIdleCharacterEnabled** (void)
  Get the idle character state.
- UeiDaqAPI void **EnableIdleCharacter** (bool enableIdleChars)
  Set the idle character state.
- UeiDaqAPI bool **IsSingleP0MarkerEnabled** (void)
  Get the single P0 marker state.
- UeiDaqAPI void **EnableSingleP0Marker** (bool enableSingleP0Marker)
  Set the single P0 marker state.
- UeiDaqAPI bool **IsTimeKeeperConnectionEnabled** (void)
  Get the time keeper "time ready" connection state.
- UeiDaqAPI void **EnableTimeKeeperConnection** (bool enableTKConnect)

Set the single P0 marker state.

- UeiDaqAPI bool **IsExtra1PPSEnabled** (void)
  Get the extra 1PPS state.

- UeiDaqAPI void **EnableExtra1PPS** (bool enableExtra1PPS)
  Set the single P0 marker state.

2.95.1 Detailed Description

Manages settings for each IRIG time code input channel

2.95.2 Member Function Documentation

2.95.2.1 UeiDaqAPI tUeiIRIGDecoderInputType UeiDaq::CUeiIRIGInputChannel::GetTimeDecoderInput ()

Get the type of time code connected to the device.

Returns:

the time decoder input type.

See also:

SetTimeDecoderInput (p. 292)

2.95.2.2 UeiDaqAPI void UeiDaq::CUeiIRIGInputChannel::SetTimeDecoderInput (tUeiIRIGDecoderInputType *input*)

Set the type of time code connected to the device.

Parameters:

input the new time decoder type.

See also:

GetTimeDecoderInput (p. 292)

2.95.2.3 UeiDaqAPI tUeiIRIGTimeCodeFormat UeiDaq::CUeiIRIGInputChannel::GetTimeCodeFormat ()

Get the time code format used by the signal connected to the device.

Returns:

the time code format.

See also:

SetTimeCodeFormat (p. 293)

2.95.2.4 UeiDaqAPI void UeiDaq::CUeiIRIGInputChannel::SetTimeCodeFormat (tUeiIRIGTimeCodeFormat *format*)

Set the time code format used by the signal connected to the device.

Parameters:

format the new time code format.

See also:

GetTimeCodeFormat (p. 292)

2.95.2.5 UeiDaqAPI bool UeiDaq::CUeiIRIGInputChannel::IsExternalClockEnabled (void)

Determines whether time decoder is using an external 10MHz clock connected to RFIn1

Returns:

The external clock state.

See also:

EnableExternalClock (p. 293)

2.95.2.6 UeiDaqAPI void UeiDaq::CUeiIRIGInputChannel::EnableExternalClock (bool *enableExternalClock*)

If selected, time decoder uses an external 10MHz clock connected to RFIn1

Parameters:

enableExternalClock true to turn external clock on, false otherwise

See also:

IsExternalClockEnabled (p. 293)

2.95.2.7 UeiDaqAPI bool UeiDaq::CUeiIRIGInputChannel::IsIdleCharacterEnabled (void)

Determines whether idle character in the timing byte stream are accepted

Returns:

The idle character state.

See also:

EnableIdleCharacter (p. 294)

2.95.2.8 UeiDaqAPI void UeiDaq::CUeiIRIGInputChannel::EnableIdleCharacter (bool *enableIdleChars*)

If selected, idle characters in the timing byte stream will be accpeted

Parameters:

enableIdleChars true to accept idle characters, false otherwise

See also:

IsIdleCharacterEnabled (p. 293)

2.95.2.9 UeiDaqAPI bool UeiDaq::CUeiIRIGInputChannel::IsSingleP0MarkerEnabled (void)

Determines whether to use only one marker P0 in the timing byte stream

Returns:

The single P0 marker state.

See also:

EnableSingleP0Marker (p. 294)

2.95.2.10 UeiDaqAPI void UeiDaq::CUeiIRIGInputChannel::EnableSingleP0Marker (bool *enableSingleP0Marker*)

If selected, Only one P0 marker is used in the timing byte stream

Parameters:

enableSingleP0Marker true to only use one P0 marker, false otherwise

See also:

IsSingleP0MarkerEnabled (p. 294)

2.95.2.11 UeiDaqAPI bool UeiDaq::CUeiIRIGInputChannel::IsTimeKeeperConnection-Enabled (void)

Determines whether "time ready" strobe is connected to time keeper

Returns:

The Time Keeper connection state.

See also:

EnableTimeKeeperConnection (p. 295)

2.95.2.12 UeiDaqAPI void UeiDaq::CUeiIRIGInputChannel::EnableTimeKeeperConnection (bool *enableTKConnect*)

If selected, "time ready" strobe is connected to time keeper

Parameters:

enableTKConnect true to keep time decoder and time keeper connected, false otherwise

See also:

IsTimeKeeperConnectionEnabled (p. 294)

2.95.2.13 UeiDaqAPI bool UeiDaq::CUeiIRIGInputChannel::IsExtra1PPSEnabled (void)

Determines whether an extra 1PPS pulse is emitted when PPM or PPH comes out of bounds

Returns:

The extra 1PPS state.

See also:

EnableExtra1PPS (p. 295)

2.95.2.14 UeiDaqAPI void UeiDaq::CUeiIRIGInputChannel::EnableExtra1PPS (bool *enableExtra1PPS*)

If selected, an extra 1PPS pulse is emitted when PPM or PPH comes out of bounds

Parameters:

enableExtra1PPS true to emit the extra 1PPS pulse, false otherwise

See also:

IsExtra1PPSEnabled (p. 295)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.96 UeiDaq::CUeiIRIGOutputChannel Class Reference

Manages settings for each IRIG time code output channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Public Member Functions

- UeiDaqAPI **CUeiIRIGOutputChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiIRIGOutputChannel** ()
  Destructor.
- UeiDaqAPI **tUeiIRIGTimeCodeFormat GetTimeCodeFormat** ()
  Get the time code format.
- UeiDaqAPI void **SetTimeCodeFormat** (**tUeiIRIGTimeCodeFormat** format)
  Set the time code format.
- UeiDaqAPI bool **IsStartWhenInputValidEnabled** (void)
  Get the start condition state.
- UeiDaqAPI void **EnableStartWhenInputValid** (bool enableStartWhenInputValid)
  Set the start condition state.

2.96.1 Detailed Description

Manages settings for each IRIG time code output channel

2.96.2 Member Function Documentation

2.96.2.1 UeiDaqAPI tUeiIRIGTimeCodeFormat UeiDaq::CUeiIRIGOutputChannel::GetTimeCodeFormat ()

Get the time code format of the signal generated by the device.

Returns:

the time code format.

See also:

SetTimeCodeFormat (p. 297)

2.96.2.2 UeiDaqAPI void UeiDaq::CUeiIRIGOutputChannel::SetTimeCodeFormat (tUeiIRIGTimeCodeFormat *format*)

Set the time code format of the signal generated by the device.

Parameters:

format the new time code format.

See also:

GetTimeCodeFormat (p. 296)

2.96.2.3 UeiDaqAPI bool UeiDaq::CUeiIRIGOutputChannel::IsStartWhenInputValid-Enabled (void)

Determines whether the output time coder waits for the input time decoder to receive a valid time code before starting

Returns:

The start confdition state.

See also:

EnableStartWhenInputValid (p. 297)

2.96.2.4 UeiDaqAPI void UeiDaq::CUeiIRIGOutputChannel::EnableStartWhenInputValid (bool *enableStartWhenInputValid*)

If selected, the output time coder waits for the input time decoder to receive a valid time code before starting

Parameters:

enableStartWhenInputValid true to wiat for input time decoder, false otherwise

See also:

IsStartWhenInputValidEnabled (p. 297)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.97 UeiDaq::CUeiIRIGReader Class Reference

IRIG Reader class.

`#include <UeiMessaging.h>`

Public Member Functions

- UeiDaqAPI **CUeiIRIGReader** (**CUeiDataStream** *pDataStream, **Int32** port=0)

  Constructor.

- virtual UeiDaqAPI ~**CUeiIRIGReader** ()

  Destructor.

- UeiDaqAPI void **Read** (**tUeiBCDTime** *pBuffer, **Int32** *status=NULL)

  Read time from the IRIG port in BCD format.

- UeiDaqAPI void **Read** (**tUeiSBSTime** *pBuffer, **Int32** *status=NULL)

  Read time from the IRIG port in SBS format.

- UeiDaqAPI void **Read** (**tUeiANSITime** *pBuffer, **Int32** *status=NULL)

  Read time from the IRIG port in C-ANSI format.

- UeiDaqAPI void **Read** (double *pBuffer, **Int32** *status=NULL)

  Read time from the IRIG port as the number of seconds since.

- UeiDaqAPI void **Read** (**Int32** numBytes, char *pBuffer, **Int32** *numBytesRead=NULL)

  Read NMEA strings from the GPS receiver associated with the IRIG port.

2.97.1 Detailed Description

This class handles reading time out of a stream coming from an IRIG interface. It returns time as BCD, SBS or C-ANSI.

2.97.2 Constructor & Destructor Documentation

2.97.2.1 UeiDaqAPI UeiDaq::CUeiIRIGReader::CUeiIRIGReader (CUeiDataStream * *pDataStream*, Int32 *port* = 0)

Parameters:

pDataStream represents the source of data to read

port the ARINC port

2.97.3 Member Function Documentation

2.97.3.1 UeiDaqAPI void UeiDaq::CUeiIRIGReader::Read (tUeiBCDTime * *pBuffer*, Int32 * *status* = NULL)

Read time from the IRIG port in Binary Coded Decimal format

Parameters:

pBuffer destination buffer

status Time keeper status

2.97.3.2 UeiDaqAPI void UeiDaq::CUeiIRIGReader::Read (tUeiSBSTime * *pBuffer*, Int32 * *status* = NULL)

Read time from the IRIG port in Straight Binary Seconds format

Parameters:

pBuffer destination buffer

status Time keeper status

2.97.3.3 UeiDaqAPI void UeiDaq::CUeiIRIGReader::Read (tUeiANSITime * *pBuffer*, Int32 * *status* = NULL)

Read time from the IRIG port in C ANSI format

Parameters:

pBuffer destination buffer

status Time keeper status

2.97.3.4 UeiDaqAPI void UeiDaq::CUeiIRIGReader::Read (double * *pBuffer*, Int32 * *status* = NULL)

Read time from the IRIG port in seconds since UNIX EPOCH 00:00:00 UTC 01/01/1970

Parameters:

pBuffer destination buffer

status Time keeper status

2.97.3.5 UeiDaqAPI void UeiDaq::CUeiIRIGReader::Read (Int32 *numBytes*, char * *pBuffer*, Int32 * *numBytesRead* = NULL)

Read NMEA string from the GPS receiver associated with the IRIG port

Parameters:

numBytes maximum number of bytes to read

pBuffer destination buffer
numBytesRead Number of bytes read from GPS receiver

The documentation for this class was generated from the following file:

- UeiMessaging.h

2.98 UeiDaq::CUeiIRIGTimeKeeperChannel Class Reference

Manages settings for each time keeper channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Public Member Functions

- UeiDaqAPI **CUeiIRIGTimeKeeperChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiIRIGTimeKeeperChannel** ()
  Destructor.
- UeiDaqAPI **tUeiIRIGTimeKeeper1PPSSource Get1PPSSource** ()
  Get the time keeper source.
- UeiDaqAPI void **Set1PPSSource** (**tUeiIRIGTimeKeeper1PPSSource** source)
  Set the time keeper source.
- UeiDaqAPI bool **IsAutoFollowEnabled** (void)
  Get the auto-follow state.
- UeiDaqAPI void **EnableAutoFollow** (bool enableAutoFollow)
  Set the auto-follow state.
- UeiDaqAPI bool **IsNominalValueEnabled** (void)
  Get the nominal value state.
- UeiDaqAPI void **EnableNominalValue** (bool enableNominalValue)
  Set the nominal value state.
- UeiDaqAPI bool **IsSubPPSEnabled** (void)
  Get the sub PPS state.
- UeiDaqAPI void **EnableSubPPS** (bool enableSubPPS)
  Set the nominal value state.
- UeiDaqAPI bool **IsSBSEnabled** (void)
  Get the SBS state.
- UeiDaqAPI void **EnableSBS** (bool enableSBS)
  Set the SBS state.
- UeiDaqAPI bool **IsInvalidSecondEnabled** (void)
  Get the invalid second state.
- UeiDaqAPI void **EnableInvalidSecond** (bool enableInvalidSecond)

Set the invalid second state.

- UeiDaqAPI bool **IsInvalidMinuteEnabled** (void)
  Get the invalid minute state.
- UeiDaqAPI void **EnableInvalidMinute** (bool enableInvalidMinute)
  Set the invalid minute state.
- UeiDaqAPI bool **IsInvalidDayEnabled** (void)
  Get the invalid day state.
- UeiDaqAPI void **EnableInvalidDay** (bool enableInvalidDay)
  Set the invalid day state.
- UeiDaqAPI **tUeiANSITime GetInitialANSITime** (void)
  Get the initial time.
- UeiDaqAPI void **SetInitialANSITime** (**tUeiANSITime** initialTime)
  Set the initial time.
- UeiDaqAPI **tUeiSBSTime GetInitialSBSTime** (void)
  Get the initial time.
- UeiDaqAPI void **SetInitialSBSTime** (**tUeiSBSTime** initialTime)
  Set the initial time.
- UeiDaqAPI bool **IsInitialTimeFromGPSEnabled** (void)
  Is initial time set from GPS.
- UeiDaqAPI void **EnableInitialTimeFromGPS** (bool setTimeFromGPS)
  Set initial time from GPS.
- UeiDaqAPI **uInt32 GetTimestampResetMask** (void)
  Get device timestamp reset mask.
- UeiDaqAPI void **SetTimestampResetMask** (**uInt32** resetMask)
  Set device timestamp reset mask.

2.98.1 Detailed Description

Manages settings for each time keeper channel

2.98.2 Member Function Documentation

2.98.2.1 UeiDaqAPI tUeiIRIGTimeKeeper1PPSSource UeiDaq::CUeiIRIGTimeKeeperChannel::Get1PPSSource ()

For proper functioning, the timekeeper requires a 1PPS signal. which can be delivered from multiple sources: internal, external, GPS... This method gets the current time keeper 1PPS source

Returns:

the 1PPS source.

See also:

Set1PPSSource (p. 303)

2.98.2.2 UeiDaqAPI void UeiDaq::CUeiIRIGTimeKeeperChannel::Set1PPSSource (tUeiIRIGTimeKeeper1PPSSource *source*)

For proper functioning, the timekeeper requires a 1PPS signal. which can be delivered from multiple sources: internal, external, GPS... This method sets the new time keeper 1PPS source

Parameters:

source the new 1 PPS source.

See also:

Get1PPSSource (p. 302)

2.98.2.3 UeiDaqAPI bool UeiDaq::CUeiIRIGTimeKeeperChannel::IsAutoFollowEnabled (void)

If selected external 1PPS source does not deliver pulses (because of a break in timecode transmission, for example). Timekeeper can switch to internal timebase when externally derived one is not available

Returns:

The auto-follow state.

See also:

EnableAutoFollow (p. 303)

2.98.2.4 UeiDaqAPI void UeiDaq::CUeiIRIGTimeKeeperChannel::EnableAutoFollow (bool *enableAutoFollow*)

If selected external 1PPS source does not deliver pulses (because of a break in timecode transmission, for example). Timekeeper can switch to internal timebase when externally derived one is not available

Parameters:

enableAutoFollow true to turn auto-follow on, false otherwise

See also:

IsAutoFollowEnabled (p. 303)

2.98.2.5 UeiDaqAPI bool UeiDaq::CUeiIRIGTimeKeeperChannel::IsNominalValueEnabled (void)

Select whether to use nominal period (i.e. 100e6 pulses of 100MHz base clock) or the period measured by timekeeper (it measures and averages number of base clock cycles between externally derived 1PPS pulses when they are valid).

Returns:

The nominal value state.

See also:

EnableNominalValue (p. 304)

2.98.2.6 UeiDaqAPI void UeiDaq::CUeiIRIGTimeKeeperChannel::EnableNominalValue (bool *enableNominalValue*)

Select whether to use nominal period (i.e. 100e6 pulses of 100MHz base clock) or the period measured by timekeeper (it measures and averages number of base clock cycles between externally derived 1PPS pulses when they are valid).

Parameters:

enableNominalValue true to turn nominal value on, false otherwise

See also:

IsNominalValueEnabled (p. 304)

2.98.2.7 UeiDaqAPI bool UeiDaq::CUeiIRIGTimeKeeperChannel::IsSubPPSEnabled (void)

Select whether external timebase is slower than 1PPS or is not derived from the timecode

Returns:

The sub PPS state.

See also:

EnableSubPPS (p. 304)

2.98.2.8 UeiDaqAPI void UeiDaq::CUeiIRIGTimeKeeperChannel::EnableSubPPS (bool *enableSubPPS*)

Select whether external timebase is slower than 1PPS or is not derived from the timecode

Parameters:

enableSubPPS true to turn sub PPS on, false otherwise

See also:

IsSubPPSEnabled (p. 304)

2.98.2.9 UeiDaqAPI bool UeiDaq::CUeiIRIGTimeKeeperChannel::IsSBSEnabled (void)

Select whether to use SBS timecode section for TimeKeeper hour/min decoding (if BCD data in the incoming timecode is corrupted)

Returns:

The SBS state.

See also:

EnableSBS (p. 305)

2.98.2.10 UeiDaqAPI void UeiDaq::CUeiIRIGTimeKeeperChannel::EnableSBS (bool *enableSBS*)

Select whether to use SBS timecode section for TimeKeeper hour/min decoding (if BCD data in the incoming timecode is corrupted)

Parameters:

enableSBS true to use SBS data instead of BCD data, false otherwise

See also:

IsSBSEnabled (p. 305)

2.98.2.11 UeiDaqAPI bool UeiDaq::CUeiIRIGTimeKeeperChannel::IsInvalidSecondEnabled (void)

Select whether seconds information is invalid in the input timecode

Returns:

The invalid second state.

See also:

EnableInvalidSecond (p. 305)

2.98.2.12 UeiDaqAPI void UeiDaq::CUeiIRIGTimeKeeperChannel::EnableInvalidSecond (bool *enableInvalidSecond*)

Select whether seconds information is invalid in the input timecode

Parameters:

enableInvalidSecond true to specify that second is invalid, false otherwise

See also:

IsInvalidSecondEnabled (p. 305)

2.98.2.13 UeiDaqAPI bool UeiDaq::CUeiIRIGTimeKeeperChannel::IsInvalidMinuteEnabled (void)

Select whether minutes information is invalid in the input timecode

Returns:

The invalid minute state.

See also:

EnableInvalidMinute (p. 306)

2.98.2.14 UeiDaqAPI void UeiDaq::CUeiIRIGTimeKeeperChannel::EnableInvalidMinute (bool *enableInvalidMinute*)

Select whether minutes information is invalid in the input timecode

Parameters:

enableInvalidMinute true to specify that minute is invalid, false otherwise

See also:

IsInvalidMinuteEnabled (p. 306)

2.98.2.15 UeiDaqAPI bool UeiDaq::CUeiIRIGTimeKeeperChannel::IsInvalidDayEnabled (void)

Select whether days information is invalid in the input timecode

Returns:

The invalid day state.

See also:

EnableInvalidDay (p. 306)

2.98.2.16 UeiDaqAPI void UeiDaq::CUeiIRIGTimeKeeperChannel::EnableInvalidDay (bool *enableInvalidDay*)

Select whether days information is invalid in the input timecode

Parameters:

enableInvalidDay true to specify that day is invalid, false otherwise

See also:

IsInvalidDayEnabled (p. 306)

2.98.2.17 UeiDaqAPI tUeiANSITime UeiDaq::CUeiIRIGTimeKeeperChannel::GetInitialANSITime (void)

Get initial time programmed in time keeper when session started

Returns:

the current value for initial time

2.98.2.18 UeiDaqAPI void UeiDaq::CUeiIRIGTimeKeeperChannel::SetInitialANSITime (tUeiANSITime *initialTime*)

Program initial time in time keeper, 1PPS signal received by timekeeper will increment the time to keep it current

Parameters:

initialTime the new value for initial time

2.98.2.19 UeiDaqAPI tUeiSBSTime UeiDaq::CUeiIRIGTimeKeeperChannel::GetInitialSBSTime (void)

Get initial time programmed in time keeper when session started

Returns:

the current value for initial time

2.98.2.20 UeiDaqAPI void UeiDaq::CUeiIRIGTimeKeeperChannel::SetInitialSBSTime (tUeiSBSTime *initialTime*)

Program initial time in time keeper, 1PPS signal received by timekeeper will increment the time to keep it current

Parameters:

initialTime the new value for initial time

2.98.2.21 UeiDaqAPI bool UeiDaq::CUeiIRIGTimeKeeperChannel::IsInitialTimeFromGPSEnabled (void)

Program initial time in time keeper from GPS. 1PPS signal received by timekeeper will increment the time to keep it current

Returns:

true if initial time is taken from GPS, false otherwise

2.98.2.22 UeiDaqAPI void UeiDaq::CUeiIRIGTimeKeeperChannel::EnableInitialTimeFromGPS (bool *setTimeFromGPS*)

Program initial time in time keeper from GPS. 1PPS signal received by timekeeper will increment the time to keep it current

Parameters:

setTimeFromGPS true to set initial time from GPS, false otherwise

2.98.2.23 UeiDaqAPI uInt32 UeiDaq::CUeiIRIGTimeKeeperChannel::GetTimestampResetMask (void)

The timestamp reset bit mask sets the layers which will have their timestamp counter reset when the IRIG session starts. The initial time is updated with the date/time of the reset.

Returns:

the current timestamp reset mask

2.98.2.24 UeiDaqAPI void UeiDaq::CUeiIRIGTimeKeeperChannel::SetTimestampResetMask (uInt32 *resetMask*)

The timestamp reset bit mask sets the layers which will have their timestamp counter reset when the IRIG session starts. The initial time is updated with the date/time of the reset.

Parameters:

resetMask the new timestamp reset mask.

The documentation for this class was generated from the following file:

- UeiChannel.h

2.99 UeiDaq::CUeiLVDTChannel Class Reference

Manages settings for each LVDT/RVDT input channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiAIChannel**.

Public Member Functions

- UeiDaqAPI **CUeiLVDTChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiLVDTChannel** ()
  Destructor.
- UeiDaqAPI **tUeiLVDTWiringScheme GetWiringScheme** ()
  Get the wiring scheme.
- UeiDaqAPI void **SetWiringScheme** (**tUeiLVDTWiringScheme** wiring)
  Set the wiring scheme.
- UeiDaqAPI bool **IsExternalExcitationEnabled** ()
  Get the external excitation state.
- UeiDaqAPI void **EnableExternalExcitation** (bool enabled)
  Enable or Disable external excitation.
- UeiDaqAPI double **GetExcitationVoltage** ()
  Get the excitation RMS voltage.
- UeiDaqAPI void **SetExcitationVoltage** (double vex)
  Set the excitation RMS voltage.
- UeiDaqAPI double **GetExcitationFrequency** ()
  Get the excitation frequency.
- UeiDaqAPI void **SetExcitationFrequency** (double fex)
  Set the excitation frequency.
- UeiDaqAPI double **GetSensorSensitivity** ()
  Get the sensor sensitivity.
- UeiDaqAPI void **SetSensorSensitivity** (double sensitivity)
  Set the sensor sensitivity.
- UeiDaqAPI double **GetFineTuningOffset** ()
  Get the fine-tuning offset.
- UeiDaqAPI void **SetFineTuningOffset** (double offset)

Set the fine-tuning offset.

- UeiDaqAPI double **GetFineTuningGain** ()
  Get the fine-tuning gain.
- UeiDaqAPI void **SetFineTuningGain** (double gain)
  Set the fine-tuning gain.

2.99.1 Detailed Description

Manages settings for each LVDT/RVDT channel

2.99.2 Member Function Documentation

2.99.2.1 UeiDaqAPI tUeiLVDTWiringScheme UeiDaq::CUeiLVDTChannel::GetWiringScheme ()

Get the current wiring scheme used for this channel.

Returns:

The current wiring scheme.

See also:

SetWiringScheme (p. 310)

2.99.2.2 UeiDaqAPI void UeiDaq::CUeiLVDTChannel::SetWiringScheme (tUeiLVDTWiringScheme *wiring*)

Specifies the wiring scheme used to connect the resistive sensor to the channel.

Parameters:

wiring The new wiring scheme.

See also:

GetWiringScheme (p. 310)

2.99.2.3 UeiDaqAPI bool UeiDaq::CUeiLVDTChannel::IsExternalExcitationEnabled ()

Determines whether the excitation will be provided by the acquisition device or by an external source

Returns:

the external excitation state.

See also:

EnableExternalExcitation (p. 311)

2.99.2.4 UeiDaqAPI void UeiDaq::CUeiLVDTChannel::EnableExternalExcitation (bool *enabled*)

Enable or disable the external excitation.

Parameters:

enabled The new external excitation state

See also:

IsExternalExcitationEnabled (p. 310)

2.99.2.5 UeiDaqAPI double UeiDaq::CUeiLVDTChannel::GetExcitationVoltage ()

Get the excitation RMS voltage configured for the channel.

Returns:

The excitation voltage.

See also:

SetExcitationVoltage (p. 311)

2.99.2.6 UeiDaqAPI void UeiDaq::CUeiLVDTChannel::SetExcitationVoltage (double *vex*)

Set the excitation RMS voltage for this channel.

Parameters:

vex The excitation voltage

See also:

GetExcitationVoltage (p. 311)

2.99.2.7 UeiDaqAPI double UeiDaq::CUeiLVDTChannel::GetExcitationFrequency ()

Get the excitation frequency configured for the channel.

Returns:

The excitation frequency.

See also:

SetExcitationFrequency (p. 312)

2.99.2.8 UeiDaqAPI void UeiDaq::CUeiLVDTChannel::SetExcitationFrequency (double *fex*)

Set the excitation frequency for this channel.

Parameters:

fex The excitation frequency

See also:

GetExcitationFrequency (p. 311)

2.99.2.9 UeiDaqAPI double UeiDaq::CUeiLVDTChannel::GetSensorSensitivity ()

Get the sensor sensitivity, it usually is in mVolts/V/[EU] Independently of the engineering unit (EU), the voltage read will be converted to mV and divided by the excitation voltage and the sensitivity

Returns:

the current sensitivity.

See also:

SetSensorSensitivity (p. 312)

2.99.2.10 UeiDaqAPI void UeiDaq::CUeiLVDTChannel::SetSensorSensitivity (double *sensitivity*)

Set the sensor sensitivity, it usually is in mVolts/V/[EU] Independently of the engineering unit (EU), the voltage read will be converted to mV and divided by the excitation voltage and the sensitivity

Parameters:

sensitivity The new sensitivity

See also:

GetSensorSensitivity (p. 312)

2.99.2.11 UeiDaqAPI double UeiDaq::CUeiLVDTChannel::GetFineTuningOffset ()

Get the fine-tuning offset

Returns:

the current offset.

See also:

SetOffset

2.99.2.12 UeiDaqAPI void UeiDaq::CUeiLVDTChannel::SetFineTuningOffset (double *offset*)

Set the fine-tuning offset

Parameters:

offset The new offset

See also:

GetOffset

2.99.2.13 UeiDaqAPI double UeiDaq::CUeiLVDTChannel::GetFineTuningGain ()

Get the fine-tuning gain

Returns:

the current fain.

See also:

SetGain (p. 54)

2.99.2.14 UeiDaqAPI void UeiDaq::CUeiLVDTChannel::SetFineTuningGain (double *gain*)

Set the fine-tuning gain

Parameters:

gain The new gain

See also:

GetGain (p. 54)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.100 UeiDaq::CUeiLVDTReader Class Reference

LVDT Reader class.

`#include <UeiReader.h>`

Public Member Functions

- UeiDaqAPI **CUeiLVDTReader** (**CUeiDataStream** *pDataStream)
  Constructor.
- virtual UeiDaqAPI ~**CUeiLVDTReader** ()
  Destructor.
- UeiDaqAPI void **ReadSingleDisplacement** (**f64** *pBuffer)
  Read displacement measured by LVDT sensor.
- UeiDaqAPI void **ReadMultipleDisplacements** (**Int32** numScans, **f64** *pBuffer)
  Read multiple displacements.
- UeiDaqAPI void **ReadCoilAmplitudes** (**f64** *pPrimaryCoilsBuffer, **f64** *pSecondaryCoilsBuffer)
  Read primary and secondary coils RMS voltages.

2.100.1 Detailed Description

Class that handles reading data of a stream coming from an LVDT input

2.100.2 Constructor & Destructor Documentation

2.100.2.1 UeiDaqAPI UeiDaq::CUeiLVDTReader::CUeiLVDTReader (CUeiDataStream * *pDataStream*)

Parameters:

pDataStream represents the source of data to read

2.100.3 Member Function Documentation

2.100.3.1 UeiDaqAPI void UeiDaq::CUeiLVDTReader::ReadSingleDisplacement (f64 * *pBuffer*)

Read one displacement scan from the input stream Ratiometric measurements are multiplied by the LVDT sensor sensitivity to obtain displacement

Parameters:

pBuffer destination buffer

2.100.3.2 UeiDaqAPI void UeiDaq::CUeiLVDTReader::ReadMultipleDisplacements (Int32 *numScans*, f64 * *pBuffer*)

Read several displacements scans from the input stream

Parameters:

numScans number of scans to read

pBuffer destination buffer

2.100.3.3 UeiDaqAPI void UeiDaq::CUeiLVDTReader::ReadCoilAmplitudes (f64 * *pPrimaryCoilsBuffer*, f64 * *pSecondaryCoilsBuffer*)

Read RMS amplitudes for the primary and secondary coils of each configured channel

Parameters:

pPrimaryCoilsBuffer destination buffer for primary coils amplitudes

pSecondaryCoilsBuffer destination buffer for secondary coils amplitudes

The documentation for this class was generated from the following file:

- UeiReader.h

2.101 UeiDaq::CUeiLVDTWriter Class Reference

LVDT Writer class.

`#include <UeiWriter.h>`

Public Member Functions

- UeiDaqAPI **CUeiLVDTWriter** (**CUeiDataStream** *pDataStream)

  Constructor.

- virtual UeiDaqAPI ~**CUeiLVDTWriter** ()

  Destructor.

- UeiDaqAPI void **WriteSingleDisplacement** (**f64** *pBuffer)

  Write displacement to simulated LVDT sensor.

- UeiDaqAPI void **WriteCoilAmplitudes** (**f64** *pPrimaryCoilsBuffer, **f64** *pSecondaryCoils-Buffer)

  Write primary coil (P1) and secondary coil (P2) amplitudes.

2.101.1 Detailed Description

Class that handles writing scaled data to a stream going to a simulated LVDT output

2.101.2 Constructor & Destructor Documentation

2.101.2.1 UeiDaqAPI UeiDaq::CUeiLVDTWriter::CUeiLVDTWriter (CUeiDataStream * *pDataStream*)

Parameters:

pDataStream represents the destination where to write data

2.101.3 Member Function Documentation

2.101.3.1 UeiDaqAPI void UeiDaq::CUeiLVDTWriter::WriteSingleDisplacement (f64 * *pBuffer*)

Write one displacement scan to the output stream Displacement is divided by the simulated LVDT sensor sensitivity to obtain a ratiometric value

Parameters:

pBuffer destination buffer

2.101.3.2 UeiDaqAPI void UeiDaq::CUeiLVDTWriter::WriteCoilAmplitudes (f64 * *pPrimaryCoilsBuffer*, f64 * *pSecondaryCoilsBuffer*)

Set both primary and secondary amplitudes as a ratiometric value. The actual amplitude depends on the excitation amplitude. Possible values must be between 0.0 (full attenuation) and 1.0 (no attenuation)

Parameters:

pPrimaryCoilsBuffer destination buffer for primary coils amplitudes

pSecondaryCoilsBuffer destination buffer for secondary coils amplitudes

The documentation for this class was generated from the following file:

- UeiWriter.h

2.102 UeiDaq::CUeiMIL1553A708ControlFrame Class Reference

CUeiMIL1553A708ControlFrame (p. 318) class - control ARINC-708 operations.

`#include <UeiFrameUtils.h>`

Inherits **_tUeiMIL1553A708ControlFrame**.

Public Member Functions

- UeiDaqAPI **CUeiMIL1553A708ControlFrame** ()
  Constructor.
- UeiDaqAPI ~**CUeiMIL1553A708ControlFrame** ()
  Destructor.

2.102.1 Detailed Description

This class is used to construct and manipulate **CUeiMIL1553A708ControlFrame** (p. 318) to simplify its use You can use a pointer to this class instead of its parent structure Class can be used only with CUeiMIL1553Writer::write()

The documentation for this class was generated from the following file:

- UeiFrameUtils.h

2.103 UeiDaq::CUeiMIL1553A708DataFrame Class Reference

CUeiMIL1553A708DataFrame (p. 319) class - exchange data with ARINC-708.

`#include <UeiFrameUtils.h>`

Inherits **_tUeiMIL1553A708DataFrame**.

Public Member Functions

- UeiDaqAPI **CUeiMIL1553A708DataFrame** ()
  Constructor.
- UeiDaqAPI ~**CUeiMIL1553A708DataFrame** ()
  Destructor.

2.103.1 Detailed Description

This class is used to construct and manipulate **CUeiMIL1553A708DataFrame** (p. 319) to simplify its use You can use a pointer to this class instead of its parent structure Class can be used only with CUeiMIL1553Writer::write()

The documentation for this class was generated from the following file:

- UeiFrameUtils.h

2.104 UeiDaq::CUeiMIL1553BCCBDataFrame Class Reference

CUeiMIL1553BCCBDataFrame (p. 320) class - assemble and store data into BC control block.

`#include <UeiFrameUtils.h>`

Inherits **_tUeiMIL1553BCCBDataFrame**.

Public Member Functions

- UeiDaqAPI **CUeiMIL1553BCCBDataFrame** ()
  Constructor.
- UeiDaqAPI **CUeiMIL1553BCCBDataFrame** (int MnFrameNum, int MnIndex, int **Block**)
  Constructor with initialization.
- UeiDaqAPI **CUeiMIL1553BCCBDataFrame** (int BCCBSegment, int BCCBIndex)
  Constructor with initialization.
- UeiDaqAPI ~**CUeiMIL1553BCCBDataFrame** ()
  Destructor.
- UeiDaqAPI void **Clear** (void)
  Clear structure Clear structure, removes all elements of data.
- UeiDaqAPI void **Set** (int MnFrameNum, int MnIndex, int **Block**)
  Clear and initialze structure with input parameters.
- UeiDaqAPI void **Set** (int BCCBSegment, int BCCBIndex)
  Clear and initialze structure with input parameters.
- UeiDaqAPI void **SetCommand** (**tUeiMIL1553CommandType** cmdType, int Rt, int Sa, int WordCount)
  Set RT-BC, BC-RT, Mode or Broadcast command.
- UeiDaqAPI void **SetCommand** (**tUeiMIL1553CommandType** cmdType, int Rt, int Sa, int WordCount, int Rt2, int Sa2)
  Set RT-RT or broadcast RT-RT command.
- UeiDaqAPI void **SetRetryOptions** (int NumRetries, **tUeiMIL1553BCRetryType** RetryType)
  Specify retry options Specify how many retries bus controller should perform until marking the entry as disabled User can select different types of reties upon encountering different fault conditions and recovery options.
- UeiDaqAPI void **SetCommandBus** (**tUeiMIL1553PortActiveBus** Bus)
  Select which bus the command is transmitted upon.
- UeiDaqAPI void **SetCommandBus** (**tUeiMIL1553PortActiveBus** enabledBus, **tUeiMIL1553PortActiveBus** initialBus)
  Select which bus the command is transmitted upon.

- UeiDaqAPI void **SetCommandDelay** (int delay_us)

  Set delay in microseconds prior to command execution.

- UeiDaqAPI void **EnableDataCompare** (int enable)

  Enable to compare received data with minimum and maximum programmed in Tmin and Tmax data arrays (one pair per data word).

- UeiDaqAPI void **EnableStatusCompare** (int number, unsigned short and_sts, unsigned short or_sts, unsigned short value, int enable)

  Enable to compare expected command status with returned one.

- UeiDaqAPI void **CopyRxData** (int Size, unsigned short *src)

  Copy transmit data into BCCB.

- UeiDaqAPI void **CopyTminData** (int Size, unsigned short *src)

  Copy minimim value to compare each received word into BCCB.

- UeiDaqAPI void **CopyTmaxData** (int Size, unsigned short *src)

  Copy maximum value to compare each received word into BCCB.

2.104.1 Detailed Description

This class is used to construct and manipulate **CUeiMIL1553BCCBDataFrame** (p. 320) to simplify its use You can use a pointer to this class instead of its parent structure Class can be used only with CUeiMIL1553Writer::write()

2.104.2 Member Function Documentation

2.104.2.1 UeiDaqAPI void UeiDaq::CUeiMIL1553BCCBDataFrame::Set (int *MnFrameNum*, int *MnIndex*, int *Block*)

Parameters:

MnFrameNum minor frame number

MnIndex entry in the minor frame

Block upper (0) or lower (1) block if this minor frame entry is involved in double-buffering operation

2.104.2.2 UeiDaqAPI void UeiDaq::CUeiMIL1553BCCBDataFrame::Set (int *BCCBSegment*, int *BCCBIndex*)

Parameters:

BCCBSegment BCCB segment to write BCCB to

BCCBIndex BCCB index to write to

2.104.2.3 UeiDaqAPI void UeiDaq::CUeiMIL1553BCCBDataFrame::SetCommand (tUeiMIL1553CommandType *cmdType*, int *Rt*, int *Sa*, int *WordCount*)

Parameters:

cmdType type of the command to transmit or receive

Rt remote terminal (31 for broadcast)

Sa subaddress (0 for mode command)

WordCount number of data words in the command

2.104.2.4 UeiDaqAPI void UeiDaq::CUeiMIL1553BCCBDataFrame::SetCommand (tUeiMIL1553CommandType *cmdType*, int *Rt*, int *Sa*, int *WordCount*, int *Rt2*, int *Sa2*)

Parameters:

cmdType type of the command to transmit or receive

Rt remote terminal (31 for broadcast)

Sa subaddress (0 for mode command)

WordCount number of data words in the command

Rt2 remote terminal (31 for broadcast)

Sa2 subaddress (0 for mode command)

2.104.2.5 UeiDaqAPI void UeiDaq::CUeiMIL1553BCCBDataFrame::SetRetryOptions (int *NumRetries*, tUeiMIL1553BCRetryType *RetryType*)

Parameters:

NumRetries number of retries to try until disabling the entry. Maximum 7 retries, 0 is to retry forever

RetryType specify failure conditions and retry options (flags can be added together)

2.104.2.6 UeiDaqAPI void UeiDaq::CUeiMIL1553BCCBDataFrame::SetCommandBus (tUeiMIL1553PortActiveBus *Bus*)

Parameters:

Bus Active bus for this command

2.104.2.7 UeiDaqAPI void UeiDaq::CUeiMIL1553BCCBDataFrame::SetCommandBus (tUeiMIL1553PortActiveBus *enabledBus*, tUeiMIL1553PortActiveBus *initialBus*)

Parameters:

enabledBus Active bus for this command

initialBus Active bus for this command

2.104.2.8 UeiDaqAPI void UeiDaq::CUeiMIL1553BCCBDataFrame::SetCommandDelay (int *delay_us*)

Parameters:

delay_us delay in microseconds

2.104.2.9 UeiDaqAPI void UeiDaq::CUeiMIL1553BCCBDataFrame::EnableStatusCompare (int *number*, unsigned short *and_sts*, unsigned short *or_sts*, unsigned short *value*, int *enable*)

Parameters:

number 0 == status1 (main status) 1 == RT-RT command status

and_sts word to perform and operation with the status before comparing it to the value

or_sts word to perform or operation with the status before comparing it to the value

value word to compare returned status with to validata command

enable 1 to enable, 0 to disable

2.104.2.10 UeiDaqAPI void UeiDaq::CUeiMIL1553BCCBDataFrame::CopyRxData (int *Size*, unsigned short * *src*)

Parameters:

Size number of words to copy

src words array

2.104.2.11 UeiDaqAPI void UeiDaq::CUeiMIL1553BCCBDataFrame::CopyTminData (int *Size*, unsigned short * *src*)

Parameters:

Size number of words to copy

src words array

2.104.2.12 UeiDaqAPI void UeiDaq::CUeiMIL1553BCCBDataFrame::CopyTmaxData (int *Size*, unsigned short * *src*)

Parameters:

Size number of words to copy

src words array

The documentation for this class was generated from the following file:

- UeiFrameUtils.h

2.105 UeiDaq::CUeiMIL1553BCCBStatusFrame Class Reference

CUeiMIL1553BCCBStatusFrame (p. 324) class - request and manipulate BC control block status.

`#include <UeiFrameUtils.h>`

Inherits **_tUeiMIL1553BCCBStatusFrame**.

Public Member Functions

- UeiDaqAPI **CUeiMIL1553BCCBStatusFrame** ()
  Constructor.
- UeiDaqAPI **CUeiMIL1553BCCBStatusFrame** (int MnFrameNum, int MnIndex, int **Block**)
  Constructor with initialization.
- UeiDaqAPI **CUeiMIL1553BCCBStatusFrame** (int BCCBSegment, int BCCBIndex)
  Constructor with initialization.
- UeiDaqAPI ~**CUeiMIL1553BCCBStatusFrame** ()
  Destructor.
- UeiDaqAPI void **Clear** (void)
  Clear structure.
- UeiDaqAPI void **Set** (int MnFrameNum, int MnIndex, int **Block**)
  Clear and initialze structure with input parameters.
- UeiDaqAPI void **Set** (int BCCBSegment, int BCCBIndex)
  Clear and initialze structure with input parameters.
- UeiDaqAPI void **GetTxData** (int Size, unsigned short *dst)
  Copy received data into user-allocated area.
- UeiDaqAPI std::string **GetBcDataStr** (int Size)
  Returns transmit command data as a string.
- UeiDaqAPI std::string **GetBus** ()
  Returns current bus as string.
- UeiDaqAPI std::string **GetRt** ()
  Returns Rt as string.
- UeiDaqAPI std::string **GetTimeStamp** ()
  Returns Rt as string.
- UeiDaqAPI std::string **GetStatusString** ()
  Returns sts1, sts2 as concatenated string.
- UeiDaqAPI std::string **GetErrorStatusString** ()
  Returns errsts0, errsts1 as concatenated string.

2.105.1 Detailed Description

This class is used to construct and manipulate **CUeiMIL1553BCCBStatusFrame** (p. 324) to simplify its use You can use a pointer to this class instead of its parent structure Class can be used only with CUeiMIL1553Reader::read()

2.105.2 Member Function Documentation

2.105.2.1 UeiDaqAPI void UeiDaq::CUeiMIL1553BCCBStatusFrame::Set (int *MnFrameNum*, int *MnIndex*, int *Block*)

Parameters:

MnFrameNum minor frame number

MnIndex entry in the minor frame

Block upper (0) or lower (1) block if this minor frame entry is involved in double-buffering operation

2.105.2.2 UeiDaqAPI void UeiDaq::CUeiMIL1553BCCBStatusFrame::Set (int *BCCBSegment*, int *BCCBIndex*)

Parameters:

BCCBSegment BCCB segment to read status from

BCCBIndex BCCB index to read status from

2.105.2.3 UeiDaqAPI void UeiDaq::CUeiMIL1553BCCBStatusFrame::GetTxData (int *Size*, unsigned short * *dst*)

Parameters:

Size number of words to copy

dst destination array

2.105.2.4 UeiDaqAPI std::string UeiDaq::CUeiMIL1553BCCBStatusFrame::GetBcDataStr (int *Size*)

Parameters:

Size number of words to print

The documentation for this class was generated from the following file:

- UeiFrameUtils.h

2.106 UeiDaq::CUeiMIL1553BCControlFrame Class Reference

CUeiMIL1553BCControlFrame (p. 326) class - control BC execution process.

`#include <UeiFrameUtils.h>`

Inherits **_tUeiMIL1553BCControlFrame**.

Public Member Functions

- UeiDaqAPI **CUeiMIL1553BCControlFrame** ()
  Constructor.
- UeiDaqAPI ~**CUeiMIL1553BCControlFrame** ()
  Destructor.
- UeiDaqAPI void **SetDisable** ()
  disable operation
- UeiDaqAPI void **SetEnableContinous** (double Minor, double Major)
  prepare frame to perform continues operation
- UeiDaqAPI void **SetOneMajorStep** ()
  prepare frame to perform one major step
- UeiDaqAPI void **SetOneMinorStep** ()
  prepare frame to perform one minor step
- UeiDaqAPI void **SetGoto** (**tUeiMIL1553BCGotoOps** command, int major, int minor)
  prepare frame to perform goto command
- UeiDaqAPI void **SelectMnBlock** (unsigned int block)
  prepare frame to select minor frame block
- UeiDaqAPI void **SwapMjEntries** ()
  enable disabled and disable enabled NJ entries with "swap" flag set

2.106.1 Detailed Description

This class is used to construct and manipulate **CUeiMIL1553BCStatusFrame** (p. 329) to simplify its use You can use a pointer to this class instead of its parent structure Class can be used only with CUeiMIL1553Writer::write()

The documentation for this class was generated from the following file:

- UeiFrameUtils.h

2.107 UeiDaq::CUeiMIL1553BCSchedFrame Class Reference

CUeiMIL1553BCSchedFrame (p. 327) class - program major and minor BC frames.

`#include <UeiFrameUtils.h>`

Inherits **_tUeiMIL1553BCSchedFrame**.

Public Member Functions

- UeiDaqAPI **CUeiMIL1553BCSchedFrame** (**tUeiMIL1553BCFrameType**)
  Constructor.
- UeiDaqAPI **CUeiMIL1553BCSchedFrame** (**tUeiMIL1553BCFrameType** SetFrameType, int MNIndex, int MNBlock, int MNOffset)
  Constructor with extra initialization.
- UeiDaqAPI **CUeiMIL1553BCSchedFrame** (**tUeiMIL1553BCFrameType** SetFrameType, int MNIndex, int MNBlock, int MNOffset, int **DataSize**)
  Constructor with extra initialization 2.
- UeiDaqAPI ~**CUeiMIL1553BCSchedFrame** ()
  Destructor.
- UeiDaqAPI void **Clear** (void)
  Clear structure.
- UeiDaqAPI void **Set** (int MNIndex, int MNBlock, int MNOffset)
  Clear and initialze structure with input parameters clear with initialization.
- UeiDaqAPI void **Set** (int MNIndex, int MNBlock, int MNOffset, int **DataSize**)
  Clear and initialze structure with input parameters (for minor frame read only) clear with initialization.
- UeiDaqAPI void **AddMajorEntry** (int MinorId, **tUeiMIL1553BCMajorFlags** entryMJ)
  Add major entry Add major entry. User can add enabled/disabled entry, execute-once entry and mark entry as a swap entry. When entry is marked as swapped every time major frame is started it incert enable bit for each swap=marked entry. This way a user can create a double-buffering mechanism IRQ?
- UeiDaqAPI void **AddMajorEntry** (int MinorId, **tUeiMIL1553BCMajorFlags** entryMJ, int BCCBSegment)
  Add major entry Add major entry. User can add enabled/disabled entry, execute-once entry and mark entry as a swap entry. When entry is marked as swapped every time major frame is started it incert enable bit for each swap=marked entry. This way a user can create a double-buffering mechanism IRQ?
- UeiDaqAPI void **AddMinorEntry** (**tUeiMIL1553BCMinorFlags** entryMN)
  Add minor entry Add minor entry. Minor entry must be enabled to be executed at least once.
- UeiDaqAPI void **AddMinorEntry** (**tUeiMIL1553BCMinorFlags** entryMN, int BCCBIndex)
  Add minor entry Add minor entry. Minor entry must be enabled to be executed at least once.

2.107.1 Detailed Description

This class is used to construct and manipulate tUeiMIL1553BCSchedFrame to simplify its use You can use a pointer to this class instead of its parent structure. Class can be used with both CUeiMIL1553Reader::read() and CUeiMIL1553Writer::write()

2.107.2 Member Function Documentation

2.107.2.1 UeiDaqAPI void UeiDaq::CUeiMIL1553BCSchedFrame::AddMajorEntry (int *MinorId*, tUeiMIL1553BCMajorFlags *entryMJ*)

Parameters:

MinorId specifies which minor ID is associated with this major entry

entryMJ specifies flags - enable, do-once and swap (flags can be added together)

2.107.2.2 UeiDaqAPI void UeiDaq::CUeiMIL1553BCSchedFrame::AddMajorEntry (int *MinorId*, tUeiMIL1553BCMajorFlags *entryMJ*, int *BCCBSegment*)

Parameters:

MinorId specifies which minor ID is associated with this major entry

entryMJ specifies flags - enable, do-once and swap (flags can be added together)

BCCBSegment - explicitly specify which BCCB segment to use

2.107.2.3 UeiDaqAPI void UeiDaq::CUeiMIL1553BCSchedFrame::AddMinorEntry (tUeiMIL1553BCMinorFlags *entryMN*)

Parameters:

entryMN specifies flags - enable and do-once (flags can be added together)

2.107.2.4 UeiDaqAPI void UeiDaq::CUeiMIL1553BCSchedFrame::AddMinorEntry (tUeiMIL1553BCMinorFlags *entryMN*, int *BCCBIndex*)

Parameters:

entryMN specifies flags - enable and do-once (flags can be added together)

BCCBIndex - explicitly specify which BCCB to use within BCCBSegment

The documentation for this class was generated from the following file:

- UeiFrameUtils.h

2.108 UeiDaq::CUeiMIL1553BCStatusFrame Class Reference

CUeiMIL1553BCStatusFrame (p. 329) class - receive BC status information.

`#include <UeiFrameUtils.h>`

Inherits **_tUeiMIL1553BCStatusFrame**.

Public Member Functions

- UeiDaqAPI **CUeiMIL1553BCStatusFrame** ()
  Constructor.
- UeiDaqAPI ~**CUeiMIL1553BCStatusFrame** ()
  Destructor.

2.108.1 Detailed Description

This class is used to construct and manipulate **CUeiMIL1553BCStatusFrame** (p. 329) to simplify its use You can use a pointer to this class instead of its parent structure Class can be used only with CUeiMIL1553Writer::read()

The documentation for this class was generated from the following file:

- UeiFrameUtils.h

2.109 UeiDaq::CUeiMIL1553BMCmdFrame Class Reference

CUeiMIL1553BMCmdFrame (p. 330) class - read BM messages divided by 1553 protocol commands.

`#include <UeiFrameUtils.h>`

Inherits **_tUeiMIL1553BMCmdFrame**.

Public Member Functions

- UeiDaqAPI **CUeiMIL1553BMCmdFrame** ()
  Constructor.
- UeiDaqAPI ~**CUeiMIL1553BMCmdFrame** ()
  Destructor.
- UeiDaqAPI void **Clear** ()
  Clear structure.
- UeiDaqAPI std::string **GetBmDataStr** ()
  Return content of the frame in the string representation.
- UeiDaqAPI std::string **GetBmDataStrDataOnly** ()
  Return data content of the frame in the string representation - no timestamp.
- UeiDaqAPI std::string **GetBmMessageStr** ()
  Return data from bus monitor in the string representation (1224-4556 for example).
- UeiDaqAPI std::string **GetTimestampStr** ()
  Return the timestamp as a string.
- UeiDaqAPI std::string **GetFullBmMessageStr** ()
  Return the entire message as a fully descriptive multi-line string.

2.109.1 Detailed Description

This class is used to construct and manipulate **CUeiMIL1553BMCmdFrame** (p. 330) to simplify its use You can use a pointer to this class instead of its parent structure Class can be used only with CUeiMIL1553Reader::read()

The documentation for this class was generated from the following file:

- UeiFrameUtils.h

2.110 UeiDaq::CUeiMIL1553BMFrame Class Reference

CUeiMIL1553BMFrame (p. 331) class - read BM messages divided by idle state of the bus.

`#include <UeiFrameUtils.h>`

Inherits **_tUeiMIL1553BMFrame**.

Public Member Functions

- UeiDaqAPI **CUeiMIL1553BMFrame** ()
  Constructor.
- UeiDaqAPI ~**CUeiMIL1553BMFrame** ()
  Destructor.
- UeiDaqAPI void **Clear** ()
  Clear structure.
- UeiDaqAPI std::string **GetBmDataStr** ()
  Return content of the frame in the string representation.
- UeiDaqAPI std::string **GetBmDataStrDataOnly** ()
  Return data content of the frame in the string representation - no timestamp.
- UeiDaqAPI std::string **GetBmMessageStr** ()
  Return data from bus monitor in the string representation (1224-4556 for example).
- UeiDaqAPI std::string **GetBus** ()
  Return Bus that this message traveled on.
- UeiDaqAPI std::string **GetAbsoluteTime** ()
  Return time in us that the current bus session has been open.
- UeiDaqAPI std::string **GetMessageGap** ()
  Return Time in us since last message.

2.110.1 Detailed Description

This class is used to construct and manipulate **CUeiMIL1553BMFrame** (p. 331) to simplify its use You can use a pointer to this class instead of its parent structure Class can be used only with CUeiMIL1553Reader::read()

The documentation for this class was generated from the following file:

- UeiFrameUtils.h

2.111 UeiDaq::CUeiMIL1553BusWriterFrame Class Reference

CUeiMIL1553BusWriterFrame (p. 332) class - write data to the bus.

`#include <UeiFrameUtils.h>`

Inherits **_tUeiMIL1553BusWriterFrame**.

Public Member Functions

- UeiDaqAPI **CUeiMIL1553BusWriterFrame** ()
  Constructor.
- UeiDaqAPI ~**CUeiMIL1553BusWriterFrame** ()
  Destructor.
- UeiDaqAPI void **Clear** ()
  Clear structure.
- UeiDaqAPI void **SetCommand** (int **Rt**, int **Sa**, int **WordCount**, **tUeiMIL1553CommandType Command**)
  Add command to Tx Frame (BC-RT or RT-BC or mode or broadcast).
- UeiDaqAPI void **SetCommand** (int **Rt**, int **Sa**, int **Rt2**, int **Sa2**, int **WordCount**, **tUeiMIL1553CommandType Command**)
  Add RT-RT command to Tx Frame.
- UeiDaqAPI void **SetDelay** (int **Delay**)
  Add pre-command delay.
- UeiDaqAPI void **CopyData** (int Size, unsigned short *src)
  Add data from the user-allocated array.

2.111.1 Detailed Description

This class is used to construct and manipulate **CUeiMIL1553BusWriterFrame** (p. 332) to simplify its use You can use a pointer to this class instead of its parent structure Class can be used only with CUeiMIL1553Writer::write()

2.111.2 Member Function Documentation

2.111.2.1 UeiDaqAPI void UeiDaq::CUeiMIL1553BusWriterFrame::CopyData (int *Size*, unsigned short * *src*)

Parameters:

Size number of words to copy

src source array

The documentation for this class was generated from the following file:

- UeiFrameUtils.h

2.112 UeiDaq::CUeiMIL1553FilterEntry Class Reference

CUeiMIL1553FilterEntry (p. 334) class - run-time control of RT.

`#include <UeiFrameUtils.h>`

Inherits **_tUeiMIL1553FilterEntry**.

Public Member Functions

- UeiDaqAPI **CUeiMIL1553FilterEntry** ()
  Constructor.
- UeiDaqAPI **CUeiMIL1553FilterEntry** (**tUeiMIL1553FilterType FilterType**, int Rt, int Sa)
  Constructor with initialization.
- UeiDaqAPI ~**CUeiMIL1553FilterEntry** ()
  Destructor.
- UeiDaqAPI void **Clear** ()
  Clear structure.
- UeiDaqAPI void **Set** (**tUeiMIL1553FilterType FilterType**, int Rt, int Sa)
  Clear and initialze structure with input parameter.
- UeiDaqAPI void **SetSizes** (int **MinRxLength**, int **MaxRxLength**, int **MinTxLength**, int **MaxTxLength**)
  Set minimum and maximum data sizes to accept by RT Set minimum and maximum data sizes to accept by RT This method requires that FilterType is set to UeiMIL1553FilterByRtSaSize If filter is not set all RTs are enabled If filter is set to UeiMIL1553FilterByRt all SAs of selected RTs are enabled If filter is set to UeiMIL1553FilterByRtSa all command and sizes of selected RT/SA are enabled.
- UeiDaqAPI void **EnableCommands** (int **EnableRxRequests**, int **EnableTxRequests**, int **EnableModeCommands**)
  Enable different types of commands This method requires that FilterType is set to UeiMIL1553FilterByRt-SaSize.

2.112.1 Detailed Description

This class is used to construct and manipulate **CUeiMIL1553FilterEntry** (p. 334) to simplify its use You can use a pointer to this class instead of its parent structure Class can be used only with CUeiMIL1553Writer::write()

2.112.2 Member Function Documentation

2.112.2.1 UeiDaqAPI void UeiDaq::CUeiMIL1553FilterEntry::Set (tUeiMIL1553FilterType *FilterType*, int *Rt*, int *Sa*)

Parameters:

FilterType select type of filtering to apply

Rt select which RT to enable

Sa select which SA to enable

2.112.2.2 UeiDaqAPI void UeiDaq::CUeiMIL1553FilterEntry::SetSizes (int *MinRxLength*, int *MaxRxLength*, int *MinTxLength*, int *MaxTxLength*)

Parameters:

MinRxLength minimum lenght for BC->RT command

MaxRxLength maximum lenght for BC->RT command

MinTxLength minimum lenght for RT->BC command

MaxTxLength maximum lenght for RT->BC command

2.112.2.3 UeiDaqAPI void UeiDaq::CUeiMIL1553FilterEntry::EnableCommands (int *EnableRxRequests*, int *EnableTxRequests*, int *EnableModeCommands*)

Parameters:

EnableRxRequests set to TRUE to enable Rx (BC->RT) requests

EnableTxRequests set to TRUE to enable Tx (RT->BC) requests

EnableModeCommands set to TRUE to enable receiving of mode commands

The documentation for this class was generated from the following file:

- UeiFrameUtils.h

2.113 UeiDaq::CUeiMIL1553Port Class Reference

Manages settings for each 1553 port.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Public Member Functions

- UeiDaqAPI **CUeiMIL1553Port** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiMIL1553Port** ()
  Destructor.
- UeiDaqAPI **tUeiMIL1553PortCoupling GetCoupling** (void)
  Get the 1553 port coupling.
- UeiDaqAPI void **SetCoupling** (**tUeiMIL1553PortCoupling** coupling)
  Set the 1553 port coupling.
- UeiDaqAPI **tUeiMIL1553PortOpMode GetPortMode** (void)
  Get the 1553 port mode of operation.
- UeiDaqAPI void **SetPortMode** (**tUeiMIL1553PortOpMode** portMode)
  Set the 1553 port mode of operation.
- UeiDaqAPI **tUeiMIL1553PortActiveBus GetTxBus** (void)
  Get the 1553 active transmission bus.
- UeiDaqAPI void **SetTxBus** (**tUeiMIL1553PortActiveBus** portBus)
  Set the 1553 active transmission bus.
- UeiDaqAPI **tUeiMIL1553PortActiveBus GetRxBus** (void)
  Get the 1553 active reception bus.
- UeiDaqAPI void **SetRxBus** (**tUeiMIL1553PortActiveBus** portBus)
  Set the 1553 active reception bus.
- UeiDaqAPI **tUeiMIL1553Endian GetRxEndian** (void)
  Get the ARINC-708 endian on reception bus Rx.
- UeiDaqAPI void **SetRxEndian** (**tUeiMIL1553Endian** endian)
  Set the ARINC-708 endian on reception bus Rx.
- UeiDaqAPI **tUeiMIL1553Endian GetTxEndian** (void)
  Get the ARINC-708 endian on transmission bus Tx.
- UeiDaqAPI void **SetTxEndian** (**tUeiMIL1553Endian** endian)

Set the ARINC-708 endian on transmission bus Tx.

- UeiDaqAPI bool **IsTimestampingEnabled** (void)
  Get the timestamping state.
- UeiDaqAPI void **EnableTimestamping** (bool enableTimestamping)
  Set the timestamping state.
- UeiDaqAPI **f64 GetTimestampResolution** ()
  Get the timestamp resolution.
- UeiDaqAPI void **SetTimestampResolution** (**f64** resolution)
  Set the timestamp resolution.
- UeiDaqAPI void **AddFilterEntry** (**tUeiMIL1553FilterEntry** entry)
  Add a filter entry.
- UeiDaqAPI **tUeiMIL1553FilterEntry** * **GetFilterEntry** (int index)
  Retrieve a filter entry.
- UeiDaqAPI void **ClearFilterEntries** (void)
  Clear filter table.
- UeiDaqAPI bool **IsFilterEnabled** (void)
  Get the filter state.
- UeiDaqAPI void **EnableFilter** (bool enableFilter)
  Set the filter state.
- UeiDaqAPI void **AddSchedulerEntry** (**tUeiMIL1553SchedulerEntry** entry)
  Add a scheduler entry.
- UeiDaqAPI **tUeiMIL1553SchedulerEntry** * **GetSchedulerEntry** (int index)
  Retrieve a scheduler entry.
- UeiDaqAPI void **ClearSchedulerEntries** (void)
  Clear scheduler table.
- UeiDaqAPI bool **IsSchedulerEnabled** (void)
  Get the scheduler state.
- UeiDaqAPI void **EnableScheduler** (bool enableScheduler)
  Set the scheduler state.
- UeiDaqAPI bool **GetInterCommandAutoDelay** (void)
  Get the auto delay state.
- UeiDaqAPI void **SetInterCommandAutoDelay** (bool enableAutoDelay)
  Set the auto delay state.

- UeiDaqAPI int **GetA708FrameSize** (void)

  Gets frame size for ARINC-708 implementation.

- UeiDaqAPI void **SetA708FrameSize** (int framesize)

  Sets frame size for ARINC-708 implementation.

- UeiDaqAPI **uInt32 GetBCOptionFlags** (void)

  Gets low-level bus controller option flags.

- UeiDaqAPI void **SetBCOptionFlags** (**uInt32** optionFlags)

  Sets low-level bus controller option flags.

2.113.1 Detailed Description

Manages settings for each 1553-553 port

2.113.2 Member Function Documentation

2.113.2.1 UeiDaqAPI tUeiMIL1553PortCoupling UeiDaq::CUeiMIL1553Port::GetCoupling (void)

Get the coupling used to to connect the port to the bus

Returns:

The 1553 port coupling.

See also:

SetCoupling (p. 338)

2.113.2.2 UeiDaqAPI void UeiDaq::CUeiMIL1553Port::SetCoupling (tUeiMIL1553PortCoupling *coupling*)

Set the coupling needed to connect port to 1553 bus

Parameters:

coupling The 1553 port coupling

See also:

GetCoupling (p. 338)

2.113.2.3 UeiDaqAPI tUeiMIL1553PortOpMode UeiDaq::CUeiMIL1553Port::GetPortMode (void)

Get the port mode of operation: BM, RT or BC

Returns:

The 1553 port mode of operation

See also:

SetPortMode (p. 339)

2.113.2.4 UeiDaqAPI void UeiDaq::CUeiMIL1553Port::SetPortMode (tUeiMIL1553PortOpMode *portMode*)

Set the port mode of operation: BM, RT or BC

Parameters:

portMode The 1553 port mode of operation

See also:

GetPortMode (p. 339)

2.113.2.5 UeiDaqAPI tUeiMIL1553PortActiveBus UeiDaq::CUeiMIL1553Port::GetTxBus (void)

Get the active bus: A or B

Returns:

The 1553 port active bus for transmission

See also:

SetTxBus (p. 339)

2.113.2.6 UeiDaqAPI void UeiDaq::CUeiMIL1553Port::SetTxBus (tUeiMIL1553PortActiveBus *portBus*)

Set the active bus: A or B

Parameters:

portBus The 1553 port active bus for transmission

See also:

GetTxBus (p. 339)

2.113.2.7 UeiDaqAPI tUeiMIL1553PortActiveBus UeiDaq::CUeiMIL1553Port::GetRxBus (void)

Get the active bus: A or B or both

Returns:

The 1553 port active bus for reception

See also:

SetRxBus (p. 340)

2.113.2.8 UeiDaqAPI void UeiDaq::CUeiMIL1553Port::SetRxBus (tUeiMIL1553PortActiveBus *portBus*)

Set the active bus: A or B or both

Parameters:

portBus The 1553 port active bus for reception

See also:

GetRxBus (p. 340)

2.113.2.9 UeiDaqAPI tUeiMIL1553Endian UeiDaq::CUeiMIL1553Port::GetRxEndian (void)

Get the endian on Rx: big or small

Returns:

The ARINC-708 endian notation on Rx

See also:

SetRxEndian (p. 340)

2.113.2.10 UeiDaqAPI void UeiDaq::CUeiMIL1553Port::SetRxEndian (tUeiMIL1553Endian *endian*)

Set the endian on Rx: big or small

Parameters:

endian The new Rx endian notation

See also:

GetRxEndian (p. 340)

2.113.2.11 UeiDaqAPI tUeiMIL1553Endian UeiDaq::CUeiMIL1553Port::GetTxEndian (void)

Get the endian on Tx: big or small

Returns:

The ARINC-708 endian notation on Tx

See also:

SetTxEndian (p. 341)

2.113.2.12 UeiDaqAPI void UeiDaq::CUeiMIL1553Port::SetTxEndian (tUeiMIL1553Endian *endian*)

Set the endian on Tx: big or small

Parameters:

endian The new Tx endian notation

See also:

GetTxEndian (p. 341)

2.113.2.13 UeiDaqAPI bool UeiDaq::CUeiMIL1553Port::IsTimestampingEnabled (void)

Determines whether each received 1553 input message is timestamped.

Returns:

The timestamping state.

See also:

EnableTimestamping (p. 341)

2.113.2.14 UeiDaqAPI void UeiDaq::CUeiMIL1553Port::EnableTimestamping (bool *enableTimestamping*)

Specifies whether each received 1553 input message is timestamped.

Parameters:

enableTimestamping true to turn timestamping on, false otherwise

See also:

IsTimestampingEnabled (p. 341)

2.113.2.15 UeiDaqAPI f64 UeiDaq::CUeiMIL1553Port::GetTimestampResolution ()

Get the timestamp resolution in seconds.

Returns:

the timestamp resolution.

See also:

SetTimestampResolution (p. 342)

2.113.2.16 UeiDaqAPI void UeiDaq::CUeiMIL1553Port::SetTimestampResolution (f64 *resolution*)

Set the timestamp resolution in seconds.

Parameters:

resolution the new resolution.

See also:

GetTimestampResolution (p. 342)

2.113.2.17 UeiDaqAPI void UeiDaq::CUeiMIL1553Port::AddFilterEntry (tUeiMIL1553FilterEntry *entry*)

Add a filter entry to the port's frame filter table. Each incoming 1553 message that doesn't match any of the filter entries is rejected.

Parameters:

entry A struture that represents the new filter entry

See also:

ClearFilterEntries (p. 343), **GetFilterEntry** (p. 342)

2.113.2.18 UeiDaqAPI tUeiMIL1553FilterEntry* UeiDaq::CUeiMIL1553Port::GetFilterEntry (int *index*)

Retrieve a filter entry from the port's frame filter table. Returns NULL when retrieving past the end of the table.

Returns:

A struture that represents the retrieved filter entry

See also:

AddFilterEntry (p. 342), **ClearFilterEntries** (p. 343)

2.113.2.19 UeiDaqAPI void UeiDaq::CUeiMIL1553Port::ClearFilterEntries (void)

Empties the port's filter table.

See also:

AddFilterEntry (p. 342), **GetFilterEntry** (p. 342)

2.113.2.20 UeiDaqAPI bool UeiDaq::CUeiMIL1553Port::IsFilterEnabled (void)

Determines whether the 1553 input filter is turned on or off.

Returns:

The filter state.

See also:

EnableFilter (p. 343)

2.113.2.21 UeiDaqAPI void UeiDaq::CUeiMIL1553Port::EnableFilter (bool *enableFilter*)

Specifies whether the 1553 input filter is turned on or off.

Parameters:

enableFilter true to turn filtering on, false otherwise

See also:

IsFilterEnabled (p. 343)

2.113.2.22 UeiDaqAPI void UeiDaq::CUeiMIL1553Port::AddSchedulerEntry (tUeiMIL1553SchedulerEntry *entry*)

Add a scheduler entry to the port's scheduler table.

Parameters:

entry A struture that represents the new scheduler entry

See also:

ClearSchedulerEntries (p. 344), **GetSchedulerEntry** (p. 343)

2.113.2.23 UeiDaqAPI tUeiMIL1553SchedulerEntry* UeiDaq::CUeiMIL1553Port::GetSchedulerEntry (int *index*)

Retrieve a scheduler entry from the port's scheduler table. Returns NULL when retrieving past the end of the table.

Returns:

A struture that represents the retrieved scheduler entry

See also:

AddSchedulerEntry (p. 343), **ClearSchedulerEntries** (p. 344)

2.113.2.24 UeiDaqAPI void UeiDaq::CUeiMIL1553Port::ClearSchedulerEntries (void)

Empties the port's scheduler table.

See also:

AddSchedulerEntry (p. 343), **GetSchedulerEntry** (p. 343)

2.113.2.25 UeiDaqAPI bool UeiDaq::CUeiMIL1553Port::IsSchedulerEnabled (void)

Determines whether scheduling is enabled.

Returns:

The scheduler state.

See also:

EnableScheduler (p. 344)

2.113.2.26 UeiDaqAPI void UeiDaq::CUeiMIL1553Port::EnableScheduler (bool *enableScheduler*)

Enables scheduling.

Parameters:

enableScheduler true to turn scheduling on, false otherwise

See also:

IsSchedulerEnabled (p. 344)

2.113.2.27 UeiDaqAPI bool UeiDaq::CUeiMIL1553Port::GetInterCommandAutoDelay (void)

Determines whether auto delay between commands on transmit is enabled.

Returns:

The auto delay settings.

See also:

SetInterCommandAutoDelay (p. 345)

2.113.2.28 UeiDaqAPI void UeiDaq::CUeiMIL1553Port::SetInterCommandAutoDelay (bool *enableAutoDelay*)

Defines whether auto delay between commands on transmit is enabled.

Parameters:

enableAutoDelay true to turn auto delay on, false otherwise

See also:

GetInterCommandAutoDelay (p. 344)

2.113.2.29 UeiDaqAPI int UeiDaq::CUeiMIL1553Port::GetA708FrameSize (void)

Determines frame size for ARINC-708 mode (in uint16s)

Returns:

The frame size

See also:

SetA708FrameSize (p. 345)

2.113.2.30 UeiDaqAPI void UeiDaq::CUeiMIL1553Port::SetA708FrameSize (int *framesize*)

Sets frame size for ARINC-708 mode (in uint16s)

Parameters:

framesize The frame size (in uint16s).

See also:

GetA708FrameSize (p. 345)

2.113.2.31 UeiDaqAPI uInt32 UeiDaq::CUeiMIL1553Port::GetBCOptionFlags (void)

Gets low-level bus controller option flags

Returns:

The current option flags

See also:

SetBCOptionFlags (p. 346)

2.113.2.32 UeiDaqAPI void UeiDaq::CUeiMIL1553Port::SetBCOptionFlags (uInt32 *optionFlags*)

Sets low-level bus controller option flags

Parameters:

optionFlags The new option flags.

See also:

GetBCOptionFlags (p. 345)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.114 UeiDaq::CUeiMIL1553Reader Class Reference

MIL-1553B Reader class.

`#include <UeiMessaging.h>`

Public Member Functions

- UeiDaqAPI **CUeiMIL1553Reader** (**CUeiDataStream** *pDataStream, **Int32** port=0)

  Constructor.

- virtual UeiDaqAPI ~**CUeiMIL1553Reader** ()

  Destructor.

- UeiDaqAPI void **Read** (**Int32** numFrames, **tUeiMIL1553Frame** *pBuffer, **Int32** *numFramesRead)

  Read tUeiMIL1553Frame(s) from the MIL1553 port.

- UeiDaqAPI void **Read** (**Int32** numFrames, **tUeiMIL1553BMFrame** *pBuffer, **Int32** *numFramesRead)

  Read tUeiMIL1553BMFrame(s) from the MIL1553 port.

- UeiDaqAPI void **Read** (**Int32** numFrames, **tUeiMIL1553BMCmdFrame** *pBuffer, **Int32** *numFramesRead)

  Read tUeiMIL1553BMCmdFrame(s) from the MIL1553 port.

- UeiDaqAPI void **Read** (**Int32** numFrames, **tUeiMIL1553RTFrame** *pBuffer, **Int32** *numFramesRead)

  Read tUeiMIL1553RTFrame(s) from the MIL1553 port.

- UeiDaqAPI void **Read** (**Int32** numFrames, **tUeiMIL1553RTStatusFrame** *pBuffer, **Int32** *numFramesRead)

  Read tUeiMIL1553RTStatusFrame(s) from the MIL1553 port.

- UeiDaqAPI void **Read** (**Int32** numFrames, **tUeiMIL1553BCCBStatusFrame** *pBuffer, **Int32** *numFramesRead)

  Read tUeiMIL1553BCCBStatusFrame(s) from the MIL1553 port.

- UeiDaqAPI void **Read** (**Int32** numFrames, **tUeiMIL1553BCSchedFrame** *pBuffer, **Int32** *numFramesRead)

  Read tUeiMIL1553BCSchedFrame(s) from the MIL1553 port.

- UeiDaqAPI void **Read** (**Int32** numFrames, **tUeiMIL1553BCStatusFrame** *pBuffer, **Int32** *numFramesRead)

  Read tUeiMIL1553BCStatusFrame(s) from the MIL1553 port.

- UeiDaqAPI void **Read** (**Int32** numFrames, **tUeiMIL1553A708DataFrame** *pBuffer, **Int32** *numFramesRead)

  Read tUeiMIL1553A708DataFrame(s) from the ARINC-708 port.

- UeiDaqAPI void **ReadAsync** (**Int32** numFrames, **tUeiMIL1553Frame** *pBuffer)

  Read frame(s) asynchronously from the MIL1553 port.

- UeiDaqAPI void **AddEventListener** (**IUeiEventListener** *pListener)

  Add an asynchronous listener.

2.114.1 Detailed Description

Class that handles reading data frames of a stream coming from an MIL-1553B interface.

2.114.2 Constructor & Destructor Documentation

2.114.2.1 UeiDaqAPI UeiDaq::CUeiMIL1553Reader::CUeiMIL1553Reader (CUeiDataStream * *pDataStream*, Int32 *port* = 0)

Parameters:

pDataStream represents the source of data to read

port the MIL1553 port

2.114.3 Member Function Documentation

2.114.3.1 UeiDaqAPI void UeiDaq::CUeiMIL1553Reader::Read (Int32 *numFrames*, tUeiMIL1553Frame * *pBuffer*, Int32 * *numFramesRead*)

Read and parse MIL1553 tUeiMIL1553Frame(s) from the input stream

Parameters:

numFrames the number of frames that can be stored in the destination buffer

pBuffer destination buffer

numFramesRead the actual number of frames read from the 1553 bus

2.114.3.2 UeiDaqAPI void UeiDaq::CUeiMIL1553Reader::Read (Int32 *numFrames*, tUeiMIL1553BMFrame * *pBuffer*, Int32 * *numFramesRead*)

Read and parse MIL1553 tUeiMIL1553BMFrame(s) from the input stream

Parameters:

numFrames the number of frames that can be stored in the destination buffer

pBuffer destination buffer

numFramesRead the actual number of frames read from the 1553 bus

2.114.3.3 UeiDaqAPI void UeiDaq::CUeiMIL1553Reader::Read (Int32 *numFrames*, tUeiMIL1553BMCmdFrame * *pBuffer*, Int32 * *numFramesRead*)

Read and parse MIL1553 tUeiMIL1553BMCmdFrame(s) from the input stream. The difference between tUeiMIL1553BMFrame and tUeiMIL1553BMCmdFrame is that the later takes care of following 1553 messaging protocol and stuffing Command[Command2][data][status][status2] into a single frame and also provides tools to convert data from the frame into string representation

Parameters:

numFrames the number of frames that can be stored in the destination buffer

pBuffer destination buffer

numFramesRead the actual number of frames read from the 1553 bus

2.114.3.4 UeiDaqAPI void UeiDaq::CUeiMIL1553Reader::Read (Int32 *numFrames*, tUeiMIL1553RTFrame * *pBuffer*, Int32 * *numFramesRead*)

Read and parse MIL1553 tUeiMIL1553RTFrame(s) from the input stream

Parameters:

numFrames the number of frames that can be stored in the destination buffer

pBuffer destination buffer

numFramesRead the actual number of frames read from the 1553 bus

2.114.3.5 UeiDaqAPI void UeiDaq::CUeiMIL1553Reader::Read (Int32 *numFrames*, tUeiMIL1553RTStatusFrame * *pBuffer*, Int32 * *numFramesRead*)

Read and parse MIL1553 tUeiMIL1553RTStatusFrame(s) from the input stream

Parameters:

numFrames the number of frames that can be stored in the destination buffer

pBuffer destination buffer

numFramesRead the actual number of frames read from the 1553 bus

2.114.3.6 UeiDaqAPI void UeiDaq::CUeiMIL1553Reader::Read (Int32 *numFrames*, tUeiMIL1553BCCBStatusFrame * *pBuffer*, Int32 * *numFramesRead*)

Read and parse MIL1553 tUeiMIL1553BCCBStatusFrame(s) from the input stream

Parameters:

numFrames the number of frames that can be stored in the destination buffer

pBuffer destination buffer

numFramesRead the actual number of frames read from the 1553 bus

2.114.3.7 UeiDaqAPI void UeiDaq::CUeiMIL1553Reader::Read (Int32 *numFrames*, tUeiMIL1553BCSchedFrame * *pBuffer*, Int32 * *numFramesRead*)

Read and parse MIL1553 tUeiMIL1553BCSchedFrame(s) from the input stream

Parameters:

numFrames the number of frames that can be stored in the destination buffer

pBuffer destination buffer

numFramesRead the actual number of frames read from the 1553 bus

2.114.3.8 UeiDaqAPI void UeiDaq::CUeiMIL1553Reader::Read (Int32 *numFrames*, tUeiMIL1553BCStatusFrame * *pBuffer*, Int32 * *numFramesRead*)

Read and parse MIL1553 tUeiMIL1553BCStatusFrame(s) from the input stream

Parameters:

numFrames the number of frames that can be stored in the destination buffer

pBuffer destination buffer

numFramesRead the actual number of frames read from the 1553 bus

2.114.3.9 UeiDaqAPI void UeiDaq::CUeiMIL1553Reader::Read (Int32 *numFrames*, tUeiMIL1553A708DataFrame * *pBuffer*, Int32 * *numFramesRead*)

Read and parse ARINC-708 tUeiMIL1553A708DataFrame(s) from the input stream

Parameters:

numFrames the number of frames that can be stored in the destination buffer

pBuffer destination buffer

numFramesRead the actual number of frames read from the ARINC-708 bus

2.114.3.10 UeiDaqAPI void UeiDaq::CUeiMIL1553Reader::ReadAsync (Int32 *numFrames*, tUeiMIL1553Frame * *pBuffer*)

Read and parse MIL1553 frames from the input stream

Parameters:

numFrames the number of frames that can be stored in the destination buffer

pBuffer destination buffer

2.114.3.11 UeiDaqAPI void UeiDaq::CUeiMIL1553Reader::AddEventListener (IUeiEventListener * *pListener*)

Subscribe a listener to receive asynchronous events

Parameters:

pListener pointer to a class that implements **IUeiEventListener** (p. 549) interface

The documentation for this class was generated from the following file:

- UeiMessaging.h

2.115 UeiDaq::CUeiMIL1553RTControlFrame Class Reference

CUeiMIL1553RTControlFrame (p. 352) class - run-time control of RT.

`#include <UeiFrameUtils.h>`

Inherits **_tUeiMIL1553RTControlFrame**.

Public Member Functions

- UeiDaqAPI **CUeiMIL1553RTControlFrame** ()
  Constructor.
- UeiDaqAPI ~**CUeiMIL1553RTControlFrame** ()
  Destructor.
- UeiDaqAPI void **Clear** ()
  Clear structure.
- UeiDaqAPI void **SetEnableMask** (int enableMask)
  Clear and initialze structure with input parameter It can be used instead of programming filter.
- UeiDaqAPI void **SetEnable** (int **Rt**, int enable)
  Clear and initialze structure with input parameter.
- UeiDaqAPI void **SelectRtBcBlock** (int **Rt**, int Block)
  Select active block for RT->BC for each Rt.
- UeiDaqAPI void **SelectBcRtBlock** (int **Rt**, int Block)
  Select active block for BC->RT for each Rt.
- UeiDaqAPI void **SetValidEntry** (int **Rt**, int **Sa**, **uInt32** validationEntry)
  Select active block for each Rt.

2.115.1 Detailed Description

This class is used to construct and manipulate **CUeiMIL1553RTControlFrame** (p. 352) to simplify its use You can use a pointer to this class instead of its parent structure Class can be used only with CUeiMIL1553Writer::write()

2.115.2 Member Function Documentation

2.115.2.1 UeiDaqAPI void UeiDaq::CUeiMIL1553RTControlFrame::SetEnableMask (int *enableMask*)

Parameters:

enableMask select which RTs are enabled (bitwise)

2.115.2.2 UeiDaqAPI void UeiDaq::CUeiMIL1553RTControlFrame::SetEnable (int *Rt*, int *enable*)

Parameters:

Rt select which RT to control

enable == TRUE enables RTs, FALSE disables them (filter permitted)

2.115.2.3 UeiDaqAPI void UeiDaq::CUeiMIL1553RTControlFrame::SelectRtBcBlock (int *Rt*, int *Block*)

Parameters:

Rt select which RT to control

Block select block either zero or one

2.115.2.4 UeiDaqAPI void UeiDaq::CUeiMIL1553RTControlFrame::SelectBcRtBlock (int *Rt*, int *Block*)

Parameters:

Rt select which RT to control

Block select block either zero or one

2.115.2.5 UeiDaqAPI void UeiDaq::CUeiMIL1553RTControlFrame::SetValidEntry (int *Rt*, int *Sa*, uInt32 *validationEntry*)

Parameters:

Rt select Rt validation entry to set for

Sa select Sa validation entry to set for

validationEntry is a validation entry

The documentation for this class was generated from the following file:

- UeiFrameUtils.h

2.116 UeiDaq::CUeiMIL1553RTFrame Class Reference

CUeiMIL1553RTFrame (p. 354) class - write or read RT "send" and "transmit" data areas.

`#include <UeiFrameUtils.h>`

Inherits **_tUeiMIL1553RTFrame**.

Public Member Functions

- UeiDaqAPI **CUeiMIL1553RTFrame** ()
  Constructor.
- UeiDaqAPI **CUeiMIL1553RTFrame** (int **Rt**, int **Sa**, int **Block**)
  Constructor with initialization.
- UeiDaqAPI **CUeiMIL1553RTFrame** (int **Rt**, int **Sa**, int **Block**, int **DataSize**)
  Constructor with initialization.
- UeiDaqAPI ~**CUeiMIL1553RTFrame** ()
  Destructor.
- UeiDaqAPI void **Clear** ()
  Clear structure.
- UeiDaqAPI void **Set** (int **Rt**, int **Sa**, int **Block**)
  Clear and initialze structure with input parameters.
- UeiDaqAPI void **Set** (int **Rt**, int **Sa**, int **Block**, int **DataSize**)
  Clear and initialze structure with input parameters.
- UeiDaqAPI void **CopyData** (int Size, unsigned short *src)
  Copy data into RT frame from the user-allocated array.
- UeiDaqAPI void **GetData** (int Size, unsigned short *dst)
  Copy fata from RT frame into user-allocated array.
- UeiDaqAPI std::string **GetFrameStr** ()
  Return content of the frame in the string representation (1-R-1-4 for example).
- UeiDaqAPI std::string **GetDataStr** ()
  Return data from RT/SA in the string representation (1224-4556 for example).

2.116.1 Detailed Description

This class is used to construct and manipulate **CUeiMIL1553RTFrame** (p. 354) to simplify its use You can use a pointer to this class instead of its parent structure Class can be used with both CUeiMIL1553Reader::read() and CUeiMIL1553Writer::write()

2.116.2 Member Function Documentation

2.116.2.1 UeiDaqAPI void UeiDaq::CUeiMIL1553RTFrame::Set (int *Rt*, int *Sa*, int *Block*)

Parameters:

Rt remote terminal number (0-31)

Sa subaddress number (0-31)

Block block number (0 or 1)

2.116.2.2 UeiDaqAPI void UeiDaq::CUeiMIL1553RTFrame::Set (int *Rt*, int *Sa*, int *Block*, int *DataSize*)

Parameters:

Rt remote terminal number (0-31)

Sa subaddress number (0-31)

Block block number (0 or 1)

DataSize number of data words to read

2.116.2.3 UeiDaqAPI void UeiDaq::CUeiMIL1553RTFrame::CopyData (int *Size*, unsigned short * *src*)

Parameters:

Size number of words to copy

src source array

2.116.2.4 UeiDaqAPI void UeiDaq::CUeiMIL1553RTFrame::GetData (int *Size*, unsigned short * *dst*)

Parameters:

Size number of words to copy

dst destination array

The documentation for this class was generated from the following file:

- UeiFrameUtils.h

2.117 UeiDaq::CUeiMIL1553RTParametersFrame Class Reference

CUeiMIL1553RTParametersFrame (p. 356) class - specify RT parameters during initialization.

`#include <UeiFrameUtils.h>`

Inherits **_tUeiMIL1553RTParametersFrame**.

Public Member Functions

- UeiDaqAPI **CUeiMIL1553RTParametersFrame** ()
  Constructor.
- UeiDaqAPI **CUeiMIL1553RTParametersFrame** (int **Rt**)
  Constructor with initialization.
- UeiDaqAPI ~**CUeiMIL1553RTParametersFrame** ()
  Destructor.
- UeiDaqAPI void **Clear** ()
  Clear structure.
- UeiDaqAPI void **Set** (int **Rt**)
  Clear and initialze structure with input parameter.

2.117.1 Detailed Description

This class is used to construct and manipulate **CUeiMIL1553RTParametersFrame** (p. 356) to simplify its use You can use a pointer to this class instead of its parent structure Class can be used only with CUeiMIL1553Writer::write()

2.117.2 Member Function Documentation

2.117.2.1 UeiDaqAPI void UeiDaq::CUeiMIL1553RTParametersFrame::Set (int *Rt*)

Parameters:

Rt select which RT to set parameters for

The documentation for this class was generated from the following file:

- UeiFrameUtils.h

2.118 UeiDaq::CUeiMIL1553RTStatusFrame Class Reference

CUeiMIL1553RTStatusFrame (p. 357) class - write or read RT "send" and "transmit" data areas.

`#include <UeiFrameUtils.h>`

Inherits **_tUeiMIL1553RTStatusFrame**.

Public Member Functions

- UeiDaqAPI **CUeiMIL1553RTStatusFrame** ()
  Constructor.
- UeiDaqAPI **CUeiMIL1553RTStatusFrame** (int **Rt**)
  Constructor with initialization.
- UeiDaqAPI ~**CUeiMIL1553RTStatusFrame** ()
  Destructor.
- UeiDaqAPI void **Clear** ()
  Clear structure.
- UeiDaqAPI void **Set** (int **Rt**)
  Clear and initialze structure with input parameter.

2.118.1 Detailed Description

This class is used to construct and manipulate **CUeiMIL1553RTStatusFrame** (p. 357) to simplify its use You can use a pointer to this class instead of its parent structure Class can be used only with CUeiMIL1553Reader::read()

2.118.2 Member Function Documentation

2.118.2.1 UeiDaqAPI void UeiDaq::CUeiMIL1553RTStatusFrame::Set (int *Rt*)

Parameters:

Rt specify what RT status to read

The documentation for this class was generated from the following file:

- UeiFrameUtils.h

2.119 UeiDaq::CUeiMIL1553Writer Class Reference

MIL-1553B Writer class.

`#include <UeiMessaging.h>`

Public Member Functions

- UeiDaqAPI **CUeiMIL1553Writer** (**CUeiDataStream** *pDataStream, **Int32** port=0)
  Constructor.
- virtual UeiDaqAPI ~**CUeiMIL1553Writer** ()
  Destructor.
- UeiDaqAPI void **Write** (**Int32** numFrames, **tUeiMIL1553Frame** *pBuffer, **Int32** *numFramesWritten)
  Write formatted tUeiMIL1553Frame(s) to the MIL1553 bus.
- UeiDaqAPI void **Write** (**Int32** numFrames, **tUeiMIL1553RTFrame** *pBuffer, **Int32** *numFramesWritten)
  Write formatted tUeiMIL1553RTFrame(s) to the MIL1553 bus.
- UeiDaqAPI void **Write** (**Int32** numFrames, **tUeiMIL1553RTControlFrame** *pBuffer, **Int32** *numFramesWritten)
  Write formatted tUeiMIL1553RTControlFrame(s) to the MIL1553 bus.
- UeiDaqAPI void **Write** (**Int32** numFrames, **tUeiMIL1553RTParametersFrame** *pBuffer, **Int32** *numFramesWritten)
  Write formatted tUeiMIL1553RTParametersFrame(s) to the MIL1553 bus.
- UeiDaqAPI void **Write** (**Int32** numFrames, **tUeiMIL1553BusWriterFrame** *pBuffer, **Int32** *numFramesWritten)
  Write formatted tUeiMIL1553BusWriterFrame(s) to the MIL1553 bus.
- UeiDaqAPI void **Write** (**Int32** numFrames, **tUeiMIL1553BCCBDataFrame** *pBuffer, **Int32** *numFramesWritten)
  Write formatted tUeiMIL1553BCCBDataFrame(s) to the MIL1553 bus.
- UeiDaqAPI void **Write** (**Int32** numFrames, **tUeiMIL1553BCSchedFrame** *pBuffer, **Int32** *numFramesWritten)
  Write formatted tUeiMIL1553BCSchedFrame(s) to the MIL1553 bus.
- UeiDaqAPI void **Write** (**Int32** numFrames, **tUeiMIL1553BCControlFrame** *pBuffer, **Int32** *numFramesWritten)
  Write formatted tUeiMIL1553BCControlFrame(s) to the MIL1553 bus.
- UeiDaqAPI void **Write** (**Int32** numFrames, **tUeiMIL1553A708ControlFrame** *pBuffer, **Int32** *numFramesWritten)
  Write formatted tUeiMIL1553A708ControlFrame(s) to the ARINC-708 bus.

- UeiDaqAPI void **Write** (**Int32** numFrames, **tUeiMIL1553A708DataFrame** *pBuffer, **Int32** *numFramesWritten)

  Write formatted tUeiMIL1553A708DataFrame(s) to the ARINC-708 bus.

- UeiDaqAPI void **WriteAsync** (**Int32** numFrames, **tUeiMIL1553Frame** *pBuffer)

  Write formatted frames(s) asynchronously to the MIL1553 bus.

- UeiDaqAPI void **AddEventListener** (**IUeiEventListener** *pListener)

  Add an asynchronous listener.

2.119.1 Detailed Description

Class that handles writing MIL-1553B frames to a stream going through an MIL-1553B interface.

2.119.2 Constructor & Destructor Documentation

2.119.2.1 UeiDaqAPI UeiDaq::CUeiMIL1553Writer::CUeiMIL1553Writer (CUeiDataStream * *pDataStream*, Int32 *port* = 0)

Parameters:

pDataStream represents the destination where to write data

port the MIL1553 port to write to

2.119.3 Member Function Documentation

2.119.3.1 UeiDaqAPI void UeiDaq::CUeiMIL1553Writer::Write (Int32 *numFrames*, tUeiMIL1553Frame * *pBuffer*, Int32 * *numFramesWritten*)

Format and send one or more tUeiMIL1553Frame(s) to the MIL1553 interface.

Parameters:

numFrames the number of frames to send

pBuffer source buffer

numFramesWritten the actual number of frames sent over the MIL1553 bus

2.119.3.2 UeiDaqAPI void UeiDaq::CUeiMIL1553Writer::Write (Int32 *numFrames*, tUeiMIL1553RTFrame * *pBuffer*, Int32 * *numFramesWritten*)

Format and send one or more tUeiMIL1553RTFrame(s) to the MIL1553 interface.

Parameters:

numFrames the number of frames to send

pBuffer source buffer

numFramesWritten the actual number of frames sent over the MIL1553 bus

2.119.3.3 UeiDaqAPI void UeiDaq::CUeiMIL1553Writer::Write (Int32 *numFrames*, tUeiMIL1553RTControlFrame * *pBuffer*, Int32 * *numFramesWritten*)

Format and send one or more tUeiMIL1553RTControlFrame(s) to the MIL1553 interface.

Parameters:

numFrames the number of frames to send

pBuffer source buffer

numFramesWritten the actual number of frames sent over the MIL1553 bus

2.119.3.4 UeiDaqAPI void UeiDaq::CUeiMIL1553Writer::Write (Int32 *numFrames*, tUeiMIL1553RTParametersFrame * *pBuffer*, Int32 * *numFramesWritten*)

Format and send one or more tUeiMIL1553RTParametersFrame(s) to the MIL1553 interface.

Parameters:

numFrames the number of frames to send

pBuffer source buffer

numFramesWritten the actual number of frames sent over the MIL1553 bus

2.119.3.5 UeiDaqAPI void UeiDaq::CUeiMIL1553Writer::Write (Int32 *numFrames*, tUeiMIL1553BusWriterFrame * *pBuffer*, Int32 * *numFramesWritten*)

Format and send one or more tUeiMIL1553BusWriterFrame(s) to the MIL1553 interface.

Parameters:

numFrames the number of frames to send

pBuffer source buffer

numFramesWritten the actual number of frames sent over the MIL1553 bus

2.119.3.6 UeiDaqAPI void UeiDaq::CUeiMIL1553Writer::Write (Int32 *numFrames*, tUeiMIL1553BCCBDataFrame * *pBuffer*, Int32 * *numFramesWritten*)

Format and send one or more tUeiMIL1553BCCBDataFrame(s) to the MIL1553 interface.

Parameters:

numFrames the number of frames to send

pBuffer source buffer

numFramesWritten the actual number of frames sent over the MIL1553 bus

2.119.3.7 UeiDaqAPI void UeiDaq::CUeiMIL1553Writer::Write (Int32 *numFrames*, tUeiMIL1553BCSchedFrame * *pBuffer*, Int32 * *numFramesWritten*)

Format and send one or more tUeiMIL1553BCSchedFrame(s) to the MIL1553 interface.

Parameters:

numFrames the number of frames to send

pBuffer source buffer

numFramesWritten the actual number of frames sent over the MIL1553 bus

2.119.3.8 UeiDaqAPI void UeiDaq::CUeiMIL1553Writer::Write (Int32 *numFrames*, tUeiMIL1553BCControlFrame * *pBuffer*, Int32 * *numFramesWritten*)

Format and send one or more tUeiMIL1553BCControlFrame(s) to the MIL1553 interface.

Parameters:

numFrames the number of frames to send

pBuffer source buffer

numFramesWritten the actual number of frames sent over the MIL1553 bus

2.119.3.9 UeiDaqAPI void UeiDaq::CUeiMIL1553Writer::Write (Int32 *numFrames*, tUeiMIL1553A708ControlFrame * *pBuffer*, Int32 * *numFramesWritten*)

Format and send one or more tUeiMIL1553A708ControlFrame(s) to the ARINC-708 interface.

Parameters:

numFrames the number of frames to send

pBuffer source buffer

numFramesWritten the actual number of frames sent over the ARINC-708 bus

2.119.3.10 UeiDaqAPI void UeiDaq::CUeiMIL1553Writer::Write (Int32 *numFrames*, tUeiMIL1553A708DataFrame * *pBuffer*, Int32 * *numFramesWritten*)

Format and send one or more tUeiMIL1553A708DataFrame(s) to the ARINC-708 interface.

Parameters:

numFrames the number of frames to send

pBuffer source buffer

numFramesWritten the actual number of frames sent over the ARINC-708 bus

2.119.3.11 UeiDaqAPI void UeiDaq::CUeiMIL1553Writer::WriteAsync (Int32 *numFrames*, tUeiMIL1553Frame * *pBuffer*)

Format and send one or more frames(s) to the MIL1553 interface.

Parameters:

numFrames the number of frames to send

pBuffer source buffer

2.119.3.12 UeiDaqAPI void UeiDaq::CUeiMIL1553Writer::AddEventListener (IUeiEventListener * *pListener*)

Subscribe a listener to receive asynchronous events

Parameters:

pListener pointer to a class that implements **IUeiEventListener** (p. 549) interface

The documentation for this class was generated from the following file:

- UeiMessaging.h

2.120 UeiDaq::CUeiMuxPort Class Reference

Manage MUX port settings.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Public Member Functions

- UeiDaqAPI **CUeiMuxPort** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiMuxPort** ()
  Destructor.
- UeiDaqAPI bool **IsBreakBeforeMakeEnabled** ()
  Get port break before make setting.
- UeiDaqAPI void **EnableBreakBeforeMake** (bool enable)
  Set port break before make setting.
- UeiDaqAPI bool **IsSyncInputEnabled** ()
  Get port synchronization input setting.
- UeiDaqAPI void **EnableSyncInput** (bool enable)
  Set port synchronization input setting.
- UeiDaqAPI bool **IsSyncInputEdgeModeEnabled** ()
  Get port synchronization input edge/level setting.
- UeiDaqAPI void **EnableSyncInputEdgeMode** (bool enable)
  Set port synchronization input edge/level setting.
- UeiDaqAPI **tUeiDigitalEdge GetSyncInputEdgePolarity** ()
  Get port synchronization input edge/level polarity.
- UeiDaqAPI void **SetSyncInputEdgePolarity** (**tUeiDigitalEdge** polarity)
  Set port synchronization input edge/level polarity.
- UeiDaqAPI **tUeiMuxSyncOutputMode GetSyncOutputMode** ()
  Get port synchronization output mode.
- UeiDaqAPI void **SetSyncOutputMode** (**tUeiMuxSyncOutputMode** mode)
  Set port synchronization output mode.
- UeiDaqAPI int **GetSyncOutputPulseWidth** ()
  Get port synchronization output pulse width.
- UeiDaqAPI void **SetSyncOutputPulseWidth** (int width)

Set port synchronization output pulse width.

- UeiDaqAPI int **GetOnDelay** ()
  Get port on delay.
- UeiDaqAPI void **SetOnDelay** (int delay)
  Set port on delay.
- UeiDaqAPI int **GetOffDelay** ()
  Get port off delay.
- UeiDaqAPI void **SetOffDelay** (int delay)
  Set port on delay.
- UeiDaqAPI void **SetHoldingVoltage** (**tUeiMuxVoltage** voltage)
  Set voltage of DC/DC during relay holding.
- UeiDaqAPI **tUeiMuxVoltage GetHoldingVoltage** ()
  Get voltage of DC/DC during relay holding.
- UeiDaqAPI void **SetSwitchingVoltage** (**tUeiMuxVoltage** voltage)
  Set voltage of DC/DC during relay switching.
- UeiDaqAPI **tUeiMuxVoltage GetSwitchingVoltage** ()
  Get voltage of DC/DC during relay switching.

2.120.1 Detailed Description

This objects stores MUX port settings

2.120.2 Member Function Documentation

2.120.2.1 UeiDaqAPI bool UeiDaq::CUeiMuxPort::IsBreakBeforeMakeEnabled ()

Get the current port break before make setting

Returns:

the current break before make

2.120.2.2 UeiDaqAPI void UeiDaq::CUeiMuxPort::EnableBreakBeforeMake (bool *enable*)

Set the port break before make setting

Parameters:

enable true to enable, false otherwise

2.120.2.3 UeiDaqAPI bool UeiDaq::CUeiMuxPort::IsSyncInputEnabled ()

Get the current port synchronization input setting A MUX device can wait for a pulse on its synchronization input before configuring its relays

Returns:

the current synchronization input

2.120.2.4 UeiDaqAPI void UeiDaq::CUeiMuxPort::EnableSyncInput (bool *enable*)

Set the port synchronization input setting A MUX device can wait for a pulse on its synchronization input before configuring its relays

Parameters:

enable true to enable, false otherwise

2.120.2.5 UeiDaqAPI bool UeiDaq::CUeiMuxPort::IsSyncInputEdgeModeEnabled ()

Get the current port synchronization input edge/level setting The sync input can work in level or edge mode

Returns:

the current synchronization input

2.120.2.6 UeiDaqAPI void UeiDaq::CUeiMuxPort::EnableSyncInputEdgeMode (bool *enable*)

Set the port synchronization input edge/level setting The sync input can work in level or edge mode

Parameters:

enable true to enable, false otherwise

2.120.2.7 UeiDaqAPI tUeiDigitalEdge UeiDaq::CUeiMuxPort::GetSyncInputEdgePolarity ()

Get the current port synchronization input edge/level polarity

Returns:

the current synchronization input polarity

2.120.2.8 UeiDaqAPI void UeiDaq::CUeiMuxPort::SetSyncInputEdgePolarity (tUeiDigitalEdge *polarity*)

Set the port synchronization input edge/level polarity

Parameters:

polarity the new polarity

2.120.2.9 UeiDaqAPI tUeiMuxSyncOutputMode UeiDaq::CUeiMuxPort::GetSyncOutputMode ()

Get the current port synchronization output mode A MUX device can emit a pulse on its synchronization output when configuring its relays

Returns:

the current synchronization output mode

2.120.2.10 UeiDaqAPI void UeiDaq::CUeiMuxPort::SetSyncOutputMode (tUeiMuxSyncOutputMode *mode*)

Set the port synchronization output mode A MUX device can emit a pulse on its synchronization output when configuring its relays

Parameters:

mode the new synchronization output mode

2.120.2.11 UeiDaqAPI int UeiDaq::CUeiMuxPort::GetSyncOutputPulseWidth ()

Get the current port synchronization output pulse width in microseconds

Returns:

the current synchronization output pulse width

2.120.2.12 UeiDaqAPI void UeiDaq::CUeiMuxPort::SetSyncOutputPulseWidth (int *width*)

Set the port synchronization output oulse width in microseconds

Parameters:

width the new synchronization output pulse width

2.120.2.13 UeiDaqAPI int UeiDaq::CUeiMuxPort::GetOnDelay ()

Get the current port on delay in microseconds

Returns:

the current on delay

2.120.2.14 UeiDaqAPI void UeiDaq::CUeiMuxPort::SetOnDelay (int *delay*)

Set the port on delay

Parameters:

delay the new on delay

2.120.2.15 UeiDaqAPI int UeiDaq::CUeiMuxPort::GetOffDelay ()

Get the current port off delay in microseconds

Returns:

the current off delay

2.120.2.16 UeiDaqAPI void UeiDaq::CUeiMuxPort::SetOffDelay (int *delay*)

Set the port off delay

Parameters:

delay the new off delay

2.120.2.17 UeiDaqAPI void UeiDaq::CUeiMuxPort::SetHoldingVoltage (tUeiMuxVoltage *voltage*)

Set voltage of DC/DC during relay holding

Parameters:

voltage the DC/DC voltage during relay holding

2.120.2.18 UeiDaqAPI tUeiMuxVoltage UeiDaq::CUeiMuxPort::GetHoldingVoltage ()

Get voltage of DC/DC during relay holding

Returns:

the current DC/DC voltage during relay holding

2.120.2.19 UeiDaqAPI void UeiDaq::CUeiMuxPort::SetSwitchingVoltage (tUeiMuxVoltage *voltage*)

Set voltage of DC/DC during relay switching

Parameters:

voltage the DC/DC voltage during relay switching

2.120.2.20 UeiDaqAPI tUeiMuxVoltage UeiDaq::CUeiMuxPort::GetSwitchingVoltage ()

Get voltage of DC/DC during relay switching

Returns:

the current DC/DC voltage during relay switching

The documentation for this class was generated from the following file:

- UeiChannel.h

2.121 UeiDaq::CUeiMuxWriter Class Reference

Mux Writer class.

`#include <UeiMessaging.h>`

Public Member Functions

- UeiDaqAPI **CUeiMuxWriter** (**CUeiDataStream** *pDataStream)
  Constructor.
- virtual UeiDaqAPI ~**CUeiMuxWriter** ()
  Destructor.
- UeiDaqAPI void **WriteRelays** (int numValues, **uInt32** *relayBuffer)
  Set relays individually.
- UeiDaqAPI void **WriteMux** (int numChannels, int *channels, int *relayIndices)
  Set mux.
- UeiDaqAPI void **WriteMuxRaw** (int numValues, **uInt32** *muxBuffer)
  Set low-level format mux values.
- UeiDaqAPI void **WriteMuxDmm** (**Int32** channelNum, **uInt32** relaySelect, **tUeiMuxDmmMode** dmmMode)
  Set single channel on MUX for use with DMM.
- UeiDaqAPI void **ReadStatus** (**uInt32** *relayA, **uInt32** *relayB, **uInt32** *relayC, **uInt32** *status)
  Get Mux status.
- UeiDaqAPI void **ReadADC** (**Int32** numVals, double *pBuffer, **Int32** *numValsRead)
  Read ADC.
- UeiDaqAPI void **ReadRelayCounts** (**Int32** countsBufferSize, **Int32** *pCountsBuffer, **Int32** *numCountsRead)
  Read current count of times each relay has been energized.

2.121.1 Detailed Description

Class that handles writing data to a stream going to a counter output

2.121.2 Constructor & Destructor Documentation

2.121.2.1 UeiDaqAPI UeiDaq::CUeiMuxWriter::CUeiMuxWriter (CUeiDataStream * *pDataStream*)

Parameters:

pDataStream represents the destination where to write data

2.121.3 Member Function Documentation

2.121.3.1 UeiDaqAPI void UeiDaq::CUeiMuxWriter::WriteRelays (int *numValues*, uInt32 * *relayBuffer*)

Set relays individually (use with care to avoid short circuits and damaging your equipment) One bit per relay [channel 0 relay A, channel 0 relay B, channel 0 relay C, channel 1 relay A, channel 1 relay B, etc...]

Parameters:

numValues Number of values to write

relayBuffer Data buffer to write

2.121.3.2 UeiDaqAPI void UeiDaq::CUeiMuxWriter::WriteMux (int *numChannels*, int * *channels*, int * *relayIndices*)

Set mux on a set of channels

Parameters:

numChannels The number of channels to set mux on

channels The list of channels to set mux on

relayIndices The index of the relay to close for each channel in the channel list (0 for relay A, 1 for relay B etc...)

2.121.3.3 UeiDaqAPI void UeiDaq::CUeiMuxWriter::WriteMuxRaw (int *numValues*, uInt32 * *muxBuffer*)

Set mux on each channel using the same representation than the low-level API Two bits per channels: 00 relays off, 01 relay A ON, 10 relay B on, 11 relay C on [channel 0 mux, channel 1 mux, channel 2 mux, etc...]

Parameters:

numValues Number of values to write

muxBuffer Data buffer to write

2.121.3.4 UeiDaqAPI void UeiDaq::CUeiMuxWriter::WriteMuxDmm (Int32 *channelNum*, uInt32 *relaySelect*, tUeiMuxDmmMode *dmmMode*)

Set mux on a single channel and select output to DMM

Parameters:

channelNum Channel to set

relaySelect Relay to enable. 0 for disable, 1 for relay A, 2 for relay B. Ignored if using UeiMuxDmmMeasRes4 dmmMode

dmmMode Select output to DMM

2.121.3.5 UeiDaqAPI void UeiDaq::CUeiMuxWriter::ReadStatus (uInt32 * *relayA*, uInt32 * *relayB*, uInt32 * *relayC*, uInt32 * *status*)

Get status of each relay on the MUX device

2.121.3.6 UeiDaqAPI void UeiDaq::CUeiMuxWriter::ReadADC (Int32 *numVals*, double * *pBuffer*, Int32 * *numValsRead*)

Read diagnostic ADC values

2.121.3.7 UeiDaqAPI void UeiDaq::CUeiMuxWriter::ReadRelayCounts (Int32 *countsBufferSize*, Int32 * *pCountsBuffer*, Int32 * *numCountsRead*)

Read current count of times each relay has been energized

Parameters:

countsBufferSize Number of relay counts to read and store in countsBuffer

pCountsBuffer Data buffer to store relay counts

numCountsRead Number of relay counts read and stored in countsBuffer

The documentation for this class was generated from the following file:

- UeiMessaging.h

2.122 UeiDaq::CUeiObject Class Reference

Base class for all objects of the UeiDaq framework.

`#include <UeiObject.h>`

Inherited by **UeiDaq::CUeiChannel**, **UeiDaq::CUeiDataStream**, **UeiDaq::CUeiDevice**, **UeiDaq::CUeiResourceParser**, **UeiDaq::CUeiSession**, **UeiDaq::CUeiTiming**, and **UeiDaq::CUeiTrigger**.

Public Member Functions

- UeiDaqAPI **CUeiObject** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiObject** ()
  Destructor.

The documentation for this class was generated from the following file:

- UeiObject.h

2.123 UeiDaq::CUeiQuadratureEncoderChannel Class Reference

Manages settings for each counter input channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiCIChannel**.

Public Member Functions

- UeiDaqAPI **CUeiQuadratureEncoderChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiQuadratureEncoderChannel** ()
  Destructor.
- UeiDaqAPI **uInt32 GetInitialPosition** ()
  Get the initial position.
- UeiDaqAPI void **SetInitialPosition** (**uInt32** initialPos)
  Set the initial position.
- UeiDaqAPI **tUeiQuadratureDecodingType GetDecodingType** ()
  Get the decoding type.
- UeiDaqAPI void **SetDecodingType** (**tUeiQuadratureDecodingType** decodingType)
  Set the decoding type.
- UeiDaqAPI bool **IsZeroIndexingEnabled** ()
  Get the zero indexing state.
- UeiDaqAPI void **EnableZeroIndexing** (bool enabled)
  Enable or Disable zero indexing.
- UeiDaqAPI **tUeiQuadratureZeroIndexPhase GetZeroIndexPhase** ()
  Get the zero indexing phase.
- UeiDaqAPI void **SetZeroIndexPhase** (**tUeiQuadratureZeroIndexPhase** zeroIndexPhase)
  Set the zero indexing phase.
- UeiDaqAPI double **GetMinimumInputAPulseWidth** ()
  Get the minimum pulse width at input A.
- UeiDaqAPI void **SetMinimumInputAPulseWidth** (double minWidth)
  Set the minimum pulse width at input A.
- UeiDaqAPI double **GetMinimumInputBPulseWidth** ()
  Get the minimum pulse width at input B.

- UeiDaqAPI void **SetMinimumInputBPulseWidth** (double minWidth)

  Set the minimum pulse width at input B.

- UeiDaqAPI double **GetMinimumInputZPulseWidth** ()

  Get the minimum pulse width at input Z.

- UeiDaqAPI void **SetMinimumInputZPulseWidth** (double minWidth)

  Set the minimum pulse width at input Z.

- UeiDaqAPI double **GetMinimumTriggerInputPulseWidth** ()

  Get the minimum pulse width at trigger input.

- UeiDaqAPI void **SetMinimumTriggerInputPulseWidth** (double minWidth)

  Set the minimum pulse width at trigger input.

- UeiDaqAPI bool **IsTimestampingEnabled** (void)

  Get the timestamping state.

- UeiDaqAPI void **EnableTimestamping** (bool enableTimestamping)

  Set the timestamping state.

- UeiDaqAPI **f64 GetTimestampResolution** ()

  Get the timestamp resolution.

- UeiDaqAPI void **SetTimestampResolution** (**f64** resolution)

  Set the timestamp resolution.

2.123.1 Detailed Description

Manages settings for each counter input channel

2.123.2 Member Function Documentation

2.123.2.1 UeiDaqAPI uInt32 UeiDaq::CUeiQuadratureEncoderChannel::GetInitialPosition ()

Get the initial number of pulses when the measurement begins. This value is reloaded each time the Z input resets the measurement.

Returns:

the initial positions

See also:

SetInitialPosition (p. 374)

2.123.2.2 UeiDaqAPI void UeiDaq::CUeiQuadratureEncoderChannel::SetInitialPosition (uInt32 *initialPos*)

Set the initial number of pulses when the measurement begins. This value is reloaded each time the Z input resets the measurement.

Parameters:

initialPos the initial position

See also:

GetInitialPosition (p. 373)

2.123.2.3 UeiDaqAPI tUeiQuadratureDecodingType UeiDaq::CUeiQuadratureEncoderChannel::GetDecodingType ()

Get the method used to count and interpret the pulses the encoder generates on input A and B. 2x and 4x decoding are more sensitive to smaller changes in position.

Returns:

the decoding type

See also:

SetDecodingType (p. 374)

2.123.2.4 UeiDaqAPI void UeiDaq::CUeiQuadratureEncoderChannel::SetDecodingType (tUeiQuadratureDecodingType *decodingType*)

Set the method used to count and interpret the pulses the encoder generates on input A and B. 2x and 4x decoding are more sensitive to smaller changes in position.

Parameters:

decodingType the decoding type

See also:

GetDecodingType (p. 374)

2.123.2.5 UeiDaqAPI bool UeiDaq::CUeiQuadratureEncoderChannel::IsZeroIndexingEnabled ()

Zero indexing resets the measurement to the initial value when the Z, A and B inputs are in a specified state.

Returns:

the zero indexing state.

See also:

EnableZeroIndexing (p. 375)

2.123.2.6 UeiDaqAPI void UeiDaq::CUeiQuadratureEncoderChannel::EnableZeroIndexing (bool *enabled*)

Zero indexing resets the measurement to the initial value when the Z, A and B inputs are in a specified state.

Parameters:

enabled the zero indexing state

See also:

IsZeroIndexingEnabled (p. 374)

2.123.2.7 UeiDaqAPI tUeiQuadratureZeroIndexPhase UeiDaq::CUeiQuadratureEncoderChannel::GetZeroIndexPhase ()

Get the states at which A and B input signals must be in when Z is high to trigger a measurement reset

Returns:

the zero indexing phase

See also:

SetZeroIndexPhase (p. 375)

2.123.2.8 UeiDaqAPI void UeiDaq::CUeiQuadratureEncoderChannel::SetZeroIndexPhase (tUeiQuadratureZeroIndexPhase *zeroIndexPhase*)

Set the states at which A and B input signals must be in when Z is high to trigger a measurement reset

Parameters:

zeroIndexPhase the zero indexing phase

See also:

GetZeroIndexPhase (p. 375)

2.123.2.9 UeiDaqAPI double UeiDaq::CUeiQuadratureEncoderChannel::GetMinimumInputAPulseWidth ()

Get the minimum pulse width in ms the counter recognizes at input A. Any pulse whose width is smaller will be ignored.

Returns:

the minimum pulse width

See also:

SetMinimumInputAPulseWidth (p. 376)

2.123.2.10 UeiDaqAPI void UeiDaq::CUeiQuadratureEncoderChannel::SetMinimumInputAPulseWidth (double *minWidth*)

Set the minimum pulse width in ms the counter recognizes at input A. Any pulse whose width is smaller will be ignored.

Parameters:

minWidth the minimum pulse width

See also:

GetMinimumInputAPulseWidth (p. 375)

2.123.2.11 UeiDaqAPI double UeiDaq::CUeiQuadratureEncoderChannel::GetMinimumInputBPulseWidth ()

Get the minimum pulse width in ms the counter recognizes at input B. Any pulse whose width is smaller will be ignored.

Returns:

the minimum pulse width

See also:

SetMinimumInputBPulseWidth (p. 376)

2.123.2.12 UeiDaqAPI void UeiDaq::CUeiQuadratureEncoderChannel::SetMinimumInputBPulseWidth (double *minWidth*)

Set the minimum pulse width in ms the counter recognizes at input B. Any pulse whose width is smaller will be ignored.

Parameters:

minWidth the minimum pulse width

See also:

GetMinimumInputBPulseWidth (p. 376)

2.123.2.13 UeiDaqAPI double UeiDaq::CUeiQuadratureEncoderChannel::GetMinimumInputZPulseWidth ()

Get the minimum pulse width in ms the counter recognizes at input Z. Any pulse whose width is smaller will be ignored.

Returns:

the minimum pulse width

See also:

SetMinimumInputZPulseWidth (p. 377)

2.123.2.14 UeiDaqAPI void UeiDaq::CUeiQuadratureEncoderChannel::SetMinimumInputZPulseWidth (double *minWidth*)

Set the minimum pulse width in ms the counter recognizes at input Z. Any pulse whose width is smaller will be ignored.

Parameters:

minWidth the minimum pulse width

See also:

GetMinimumInputZPulseWidth (p. 376)

2.123.2.15 UeiDaqAPI double UeiDaq::CUeiQuadratureEncoderChannel::GetMinimumTriggerInputPulseWidth ()

Get the minimum pulse width in ms the counter recognizes at trigger input. Any pulse whose width is smaller will be ignored.

Returns:

the minimum pulse width

See also:

SetMinimumTriggerInputPulseWidth (p. 377)

2.123.2.16 UeiDaqAPI void UeiDaq::CUeiQuadratureEncoderChannel::SetMinimumTriggerInputPulseWidth (double *minWidth*)

Set the minimum pulse width in ms the counter recognizes at trigger input. Any pulse whose width is smaller will be ignored.

Parameters:

minWidth the minimum pulse width

See also:

GetMinimumTriggerInputPulseWidth (p. 377)

2.123.2.17 UeiDaqAPI bool UeiDaq::CUeiQuadratureEncoderChannel::IsTimestampingEnabled (void)

Determines whether each received measurement is timestamped.

Returns:

The timestamping state.

See also:

EnableTimestamping (p. 378)

2.123.2.18 UeiDaqAPI void UeiDaq::CUeiQuadratureEncoderChannel::EnableTimestamping (bool *enableTimestamping*)

Specifies whether each received measurement is timestamped.

Parameters:

enableTimestamping true to turn timestamping on, false otherwise

See also:

IsTimestampingEnabled (p. 377)

2.123.2.19 UeiDaqAPI f64 UeiDaq::CUeiQuadratureEncoderChannel::GetTimestampResolution ()

Get the timestamp resolution in seconds.

Returns:

the timestamp resolution.

See also:

SetTimestampResolution (p. 378)

2.123.2.20 UeiDaqAPI void UeiDaq::CUeiQuadratureEncoderChannel::SetTimestampResolution (f64 *resolution*)

Set the timestamp resolution in seconds.

Parameters:

resolution the new resolution.

See also:

GetTimestampResolution (p. 378)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.124 UeiDaq::CUeiResistanceChannel Class Reference

Manages settings for each resistance input channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiAIChannel**.

Inherited by **UeiDaq::CUeiRTDChannel**.

Public Member Functions

- UeiDaqAPI **CUeiResistanceChannel** ()

  Constructor.

- virtual UeiDaqAPI ~**CUeiResistanceChannel** ()

  Destructor.

- UeiDaqAPI **tUeiWiringScheme GetWiringScheme** ()

  Get the wiring scheme.

- UeiDaqAPI void **SetWiringScheme** (**tUeiWiringScheme** wiring)

  Set the wiring scheme.

- UeiDaqAPI double **GetExcitationVoltage** ()

  Get the excitation voltage.

- UeiDaqAPI void **SetExcitationVoltage** (double vex)

  Set the excitation voltage.

- UeiDaqAPI **tUeiReferenceResistorType GetReferenceResistorType** ()

  Get the reference resistor type.

- UeiDaqAPI void **SetReferenceResistorType** (**tUeiReferenceResistorType** refResistorType)

  Set the reference resistor type.

- UeiDaqAPI double **GetReferenceResistance** ()

  Get the reference resistance.

- UeiDaqAPI void **SetReferenceResistance** (double rref)

  Set the reference resistance.

2.124.1 Detailed Description

Manages settings for each resistance input channel

2.124.2 Member Function Documentation

2.124.2.1 UeiDaqAPI tUeiWiringScheme UeiDaq::CUeiResistanceChannel::GetWiringScheme ()

Get the current wiring scheme used for this channel.

Returns:

The current wiring scheme.

See also:

SetWiringScheme (p. 380)

2.124.2.2 UeiDaqAPI void UeiDaq::CUeiResistanceChannel::SetWiringScheme (tUeiWiringScheme *wiring*)

Specifies the wiring scheme used to connect the resistive sensor to the channel.

Parameters:

wiring The new wiring scheme.

See also:

GetWiringScheme (p. 380)

2.124.2.3 UeiDaqAPI double UeiDaq::CUeiResistanceChannel::GetExcitationVoltage ()

Get the excitation voltage configured for the channel.

Returns:

The excitation voltage.

See also:

SetExcitationVoltage (p. 380)

2.124.2.4 UeiDaqAPI void UeiDaq::CUeiResistanceChannel::SetExcitationVoltage (double *vex*)

Set the excitation voltage for this channel.

Parameters:

vex The excitation voltage

See also:

GetExcitationVoltage (p. 380)

2.124.2.5 UeiDaqAPI tUeiReferenceResistorType UeiDaq::CUeiResistanceChannel::GetReferenceResistorType ()

The reference resistor can be built-in the connector block if you are using a DNA-STP-AIU or connected externally.

Returns:

The reference resistor type.

See also:

SetReferenceResistorType (p. 381)

2.124.2.6 UeiDaqAPI void UeiDaq::CUeiResistanceChannel::SetReferenceResistorType (tUeiReferenceResistorType *refResistorType*)

The reference resistor can be built-in the connector block if you are using a DNA-STP-AIU or connected externally.

Parameters:

refResistorType The reference resistor type.

See also:

GetReferenceResistorType (p. 381)

2.124.2.7 UeiDaqAPI double UeiDaq::CUeiResistanceChannel::GetReferenceResistance ()

Get the reference resistance used to measure the current flowing through the resistive sensor.

Returns:

The reference resistance.

See also:

SetReferenceResistance (p. 381)

2.124.2.8 UeiDaqAPI void UeiDaq::CUeiResistanceChannel::SetReferenceResistance (double *rref*)

Set the reference resistance used to measure the current flowing through the resistive sensor.

Parameters:

rref The reference resistance

See also:

GetReferenceResistance (p. 381)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.125 UeiDaq::CUeiResourceParser Class Reference

Resource string parser class.

`#include <UeiResourceParser.h>`

Inherits **UeiDaq::CUeiObject**.

Public Member Functions

- UeiDaqAPI **CUeiResourceParser** (std::string resource)
  Constructor.
- virtual UeiDaqAPI ~**CUeiResourceParser** ()
  Destructor.
- UeiDaqAPI std::string **GetDeviceClass** ()
  Get device class name.
- UeiDaqAPI **tUeiSessionType GetSessionType** ()
  Get session type.
- UeiDaqAPI std::list< int > **GetChannelList** ()
  Get channel list.
- UeiDaqAPI int **GetDeviceID** ()
  Get the device id.
- UeiDaqAPI unsigned int **GetRemoteAddressU32** ()
  Get the remote address.
- UeiDaqAPI std::string **GetRemoteAddress** ()
  Get the remote address.

2.125.1 Detailed Description

This class parses a resource string and gives access to the resource components: -Device Class -Device IP Address -Device ID -Channel List

2.125.2 Member Function Documentation

2.125.2.1 UeiDaqAPI std::string UeiDaq::CUeiResourceParser::GetDeviceClass ()

Get the device class name specified in the resource string [devclass]://[IP address]/Dev[ID]/[SessionType][Channel List]

Returns:

A string that contains the device class name.

2.125.2.2 UeiDaqAPI tUeiSessionType UeiDaq::CUeiResourceParser::GetSessionType ()

Get the session type specified in the resource string [devclass]://[IP address]/Dev[ID]/[SessionType][Channel List]

Returns:

The session type.

2.125.2.3 UeiDaqAPI std::list<int> UeiDaq::CUeiResourceParser::GetChannelList ()

Get the channel list specified in the resource string [devclass]://[IP address]/Dev[ID]/[SessionType][Channel List]

Returns:

A list of integer values representing each channel.

2.125.2.4 UeiDaqAPI int UeiDaq::CUeiResourceParser::GetDeviceID ()

Get the device id specified in the resource string [devclass]://[IP address]/Dev[ID]/[SessionType][Channel List]

Returns:

A number representing the device id.

2.125.2.5 UeiDaqAPI unsigned int UeiDaq::CUeiResourceParser::GetRemoteAddressU32 ()

Get the remote address specified in the resource string [devclass]://[IP address]/Dev[ID]/[SessionType][Channel List]

Returns:

The address as an integer.

2.125.2.6 UeiDaqAPI std::string UeiDaq::CUeiResourceParser::GetRemoteAddress ()

Get the remote address specified in the resource string [devclass]://[IP address]/Dev[ID]/[SessionType][Channel List]

Returns:

The address as a string.

The documentation for this class was generated from the following file:

- UeiResourceParser.h

2.126 UeiDaq::CUeiRTDChannel Class Reference

Manages settings for each RTD input channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiResistanceChannel**.

Public Member Functions

- UeiDaqAPI **CUeiRTDChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiRTDChannel** ()
  Destructor.
- UeiDaqAPI **tUeiRTDType GetRTDType** ()
  Get the RTD type.
- UeiDaqAPI void **SetRTDType** (**tUeiRTDType** type)
  Set the RTD type.
- UeiDaqAPI double **GetRTDNominalResistance** ()
  Get the RTD nominal resistance.
- UeiDaqAPI void **SetRTDNominalResistance** (double r0)
  Set the RTD nominal resistance.
- UeiDaqAPI double **GetCoefficientA** ()
  Get the Callendar Van-Dusen coefficient A.
- UeiDaqAPI void **SetCoefficientA** (double A)
  Set the Callendar Van-Dusen coefficient A.
- UeiDaqAPI double **GetCoefficientB** ()
  Get the Callendar Van-Dusen coefficient B.
- UeiDaqAPI void **SetCoefficientB** (double B)
  Set the Callendar Van-Dusen coefficient B.
- UeiDaqAPI double **GetCoefficientC** ()
  Get the Callendar Van-Dusen coefficient C.
- UeiDaqAPI void **SetCoefficientC** (double C)
  Set the Callendar Van-Dusen coefficient C.
- UeiDaqAPI **tUeiTemperatureScale GetTemperatureScale** ()
  Get the temperature scale.
- UeiDaqAPI void **SetTemperatureScale** (**tUeiTemperatureScale** tempScale)
  Set the temperature scale.

2.126.1 Detailed Description

Manages settings for each RTD input channel

2.126.2 Member Function Documentation

2.126.2.1 UeiDaqAPI tUeiRTDType UeiDaq::CUeiRTDChannel::GetRTDType ()

RTD sensors are specified using the "alpha" (a) constant It is also known as the temperature coefficient of resistance, and symbolizes the resistance change factor per degree of temperature change. The RTD type is used to select the proper coefficients A, B and C for the Callendar Van-Dusen equation used to convert resistance measurements to temperature.

Returns:

The RTD type.

See also:

SetRTDType (p. 385)

2.126.2.2 UeiDaqAPI void UeiDaq::CUeiRTDChannel::SetRTDType (tUeiRTDType *type*)

RTD sensors are specified using the "alpha" (a) constant It is also known as the temperature coefficient of resistance, and symbolizes the resistance change factor per degree of temperature change. The RTD type is used to select the proper coefficients A, B and C for the Callendar Van-Dusen equation used to convert resistance measurements to temperature.

Parameters:

type The RTD type

See also:

GetRTDType (p. 385)

2.126.2.3 UeiDaqAPI double UeiDaq::CUeiRTDChannel::GetRTDNominalResistance ()

Get the RTD nominal resistance at 0 deg.

Returns:

The RTD nominal resistance.

See also:

SetRTDNominalResistance (p. 386)

2.126.2.4 UeiDaqAPI void UeiDaq::CUeiRTDChannel::SetRTDNominalResistance (double *r0*)

Set the RTD nominal resistance at 0 deg.

Parameters:

r0 The RTD nominal resistance

See also:

GetRTDNominalResistance (p. 385)

2.126.2.5 UeiDaqAPI double UeiDaq::CUeiRTDChannel::GetCoefficientA ()

The Callendar-Van Dusen equation is an equation that describes the relationship between resistance (R) and temperature (t) of platinum resistance thermometers. t < 0: R(t) = R(0)[1 + A * t + B * t * t + (t - 100)C * t * t * t]. t > 0: R(t) = R(0)(1 + A * t + B * t * t).

Returns:

The CVD A coefficient.

See also:

SetCoefficientA (p. 386)

2.126.2.6 UeiDaqAPI void UeiDaq::CUeiRTDChannel::SetCoefficientA (double *A*)

The Callendar-Van Dusen equation is an equation that describes the relationship between resistance (R) and temperature (t) of platinum resistance thermometers. t < 0: R(t) = R(0)[1 + A * t + B * t * t + (t - 100)C * t * t * t]. t > 0: R(t) = R(0)(1 + A * t + B * t * t).

Parameters:

A The CVD A coefficient

See also:

GetCoefficientA (p. 386)

2.126.2.7 UeiDaqAPI double UeiDaq::CUeiRTDChannel::GetCoefficientB ()

The Callendar-Van Dusen equation is an equation that describes the relationship between resistance (R) and temperature (t) of platinum resistance thermometers. t < 0: R(t) = R(0)[1 + A * t + B * t * t + (t - 100)C * t * t * t]. t > 0: R(t) = R(0)(1 + A * t + B * t * t).

Returns:

The CVD B coefficient.

See also:

SetCoefficientB (p. 387)

2.126.2.8 UeiDaqAPI void UeiDaq::CUeiRTDChannel::SetCoefficientB (double *B*)

The Callendar-Van Dusen equation is an equation that describes the relationship between resistance (R) and temperature (t) of platinum resistance thermometers. t < 0: R(t) = R(0)[1 + A * t + B * t * t + (t - 100)C * t * t * t]. t > 0: R(t) = R(0)(1 + A * t + B * t * t).

Parameters:

B The CVD B coefficient

See also:

GetCoefficientB (p. 386)

2.126.2.9 UeiDaqAPI double UeiDaq::CUeiRTDChannel::GetCoefficientC ()

The Callendar-Van Dusen equation is an equation that describes the relationship between resistance (R) and temperature (t) of platinum resistance thermometers. t < 0: R(t) = R(0)[1 + A * t + B * t * t + (t - 100)C * t * t * t]. t > 0: R(t) = R(0)(1 + A * t + B * t * t).

Returns:

The CVD C coefficient.

See also:

SetCoefficientC (p. 387)

2.126.2.10 UeiDaqAPI void UeiDaq::CUeiRTDChannel::SetCoefficientC (double *C*)

The Callendar-Van Dusen equation is an equation that describes the relationship between resistance (R) and temperature (t) of platinum resistance thermometers. t < 0: R(t) = R(0)[1 + A * t + B * t * t + (t - 100)C * t * t * t]. t > 0: R(t) = R(0)(1 + A * t + B * t * t).

Parameters:

C The CVD C coefficient

See also:

GetCoefficientC (p. 387)

2.126.2.11 UeiDaqAPI tUeiTemperatureScale UeiDaq::CUeiRTDChannel::GetTemperatureScale ()

Get the temperature scale used to convert the measured resistance to temperature.

Returns:

the temperature scale.

See also:

SetTemperatureScale (p. 388)

2.126.2.12 UeiDaqAPI void UeiDaq::CUeiRTDChannel::SetTemperatureScale (tUeiTemperatureScale *tempScale*)

Set the temperature scale used to convert the measured resistance to temperature.

Parameters:

tempScale the temperature scale.

See also:

GetTemperatureScale (p. 387)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.127 UeiDaq::CUeiSerialPort Class Reference

Manages settings for each serial port.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Public Member Functions

- UeiDaqAPI **CUeiSerialPort** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiSerialPort** ()
  Destructor.
- UeiDaqAPI **tUeiSerialPortMode GetMode** ()
  Get the serial port mode.
- UeiDaqAPI void **SetMode** (**tUeiSerialPortMode** mode)
  Set the serial port mode.
- UeiDaqAPI **tUeiSerialPortSpeed GetSpeed** ()
  Get the serial port speed.
- UeiDaqAPI void **SetSpeed** (**tUeiSerialPortSpeed** bitsPerSecond)
  Set the serial port speed.
- UeiDaqAPI **uInt32 GetCustomSpeed** ()
  Get the serial port custom speed.
- UeiDaqAPI void **SetCustomSpeed** (**uInt32** bitsPerSecond)
  Set the serial port custom speed.
- UeiDaqAPI **tUeiSerialPortDataBits GetDataBits** ()
  Get the serial port data bits.
- UeiDaqAPI void **SetDataBits** (**tUeiSerialPortDataBits** dataBits)
  Set the serial port data bits.
- UeiDaqAPI **tUeiSerialPortParity GetParity** ()
  Get the serial port parity.
- UeiDaqAPI void **SetParity** (**tUeiSerialPortParity** parity)
  Set the serial port parity.
- UeiDaqAPI **tUeiSerialPortStopBits GetStopBits** ()
  Get the serial port stop bits.
- UeiDaqAPI void **SetStopBits** (**tUeiSerialPortStopBits** stopBits)

Set the serial port stop bits.

- UeiDaqAPI std::string **GetTermination** (void)
  Get the "end of line" character sequence.
- UeiDaqAPI void **SetTermination** (std::string eol)
  Set the "end of line" character sequence.
- UeiDaqAPI bool **IsTxTerminationResistorEnabled** ()
  Get the TX termination resistor state.
- UeiDaqAPI void **EnableTxTerminationResistor** (bool enabled)
  Enable or disable TX termination resistor.
- UeiDaqAPI bool **IsRxTerminationResistorEnabled** ()
  Get the RX termination resistor state.
- UeiDaqAPI void **EnableRxTerminationResistor** (bool enabled)
  Enable or disable RX termination resistor.
- UeiDaqAPI bool **IsErrorReportingEnabled** ()
  Get the error reporting state.
- UeiDaqAPI void **EnableErrorReporting** (bool enabled)
  Enable or disable error reporting.
- UeiDaqAPI **tUeiSerialPortFlowControl GetFlowControl** ()
  Get Flow control.
- UeiDaqAPI void **SetFlowControl** (**tUeiSerialPortFlowControl** flowControl)
  Set Flow control.
- UeiDaqAPI bool **IsHDEchoSuppressionEnabled** ()
  Get the half-duplex echo suppression state.
- UeiDaqAPI void **EnableHDEchoSuppression** (bool enabled)
  Enable or disable half-duplex echo suppression.
- UeiDaqAPI bool **IsTxAutoDisableEnabled** ()
  Get the TX auto-disable state.
- UeiDaqAPI void **EnableTxAutoDisable** (bool enabled)
  Enable or disable TX auto-disable.
- UeiDaqAPI bool **IsOnTheFlyParityBitEnabled** ()
  Get the on the fly parity bit state.
- UeiDaqAPI void **EnableOnTheFlyParityBit** (bool enabled)
  Enable or disable on the fly parity bit.

- UeiDaqAPI bool **IsTimeoutUponReceiveEnabled** ()
  Get timeout error upon receiving data state.
- UeiDaqAPI void **EnableTimeoutUponReceive** (bool enabled)
  Enable or disable timeout error upon reading data.
- UeiDaqAPI **uInt32 GetCharDelay** ()
  Get character delay in microseconds.
- UeiDaqAPI void **SetCharDelay** (**uInt32** charDelayUs)
  Set character delay in microseconds.
- UeiDaqAPI **tUeiSerialPortMinorFrameMode GetMinorFrameMode** ()
  Get minor frame mode.
- UeiDaqAPI void **SetMinorFrameMode** (**tUeiSerialPortMinorFrameMode** mode)
  Set minor frame mode.
- UeiDaqAPI **uInt32 GetMinorFrameLength** ()
  Get minor frame length.
- UeiDaqAPI void **SetMinorFrameLength** (**uInt32** length)
  Set minor frame length.
- UeiDaqAPI **uInt32 GetMinorFrameDelay** ()
  Get minor frame delay in microseconds.
- UeiDaqAPI void **SetMinorFrameDelay** (**uInt32** frameDelayUs)
  Set minor frame delay in microseconds.
- UeiDaqAPI **uInt32 GetMajorFramePeriod** ()
  Get major frame period in microseconds.
- UeiDaqAPI void **SetMajorFramePeriod** (**uInt32** framePeriodUs)
  Set major frame period in microseconds.
- UeiDaqAPI **f64 GetTimestampResolution** ()
  Get the timestamp resolution.
- UeiDaqAPI void **SetTimestampResolution** (**f64** resolution)
  Set the timestamp resolution.
- UeiDaqAPI bool **IsLoopbackEnabled** ()
  Get the loopback mode state.
- UeiDaqAPI void **EnableLoopback** (bool enabled)
  Enable or disable half-duplex echo suppression.
- UeiDaqAPI bool **IsBreakEnabled** ()
  Get break detection state.

- UeiDaqAPI void **EnableBreak** (bool enabled)
  Enable or disable break detection.
- UeiDaqAPI char **GetReceiveBreakCharacter** ()
  Get the receive break character.
- UeiDaqAPI void **SetReceiveBreakCharacter** (char breakChar)
  Set the receive break character.

2.127.1 Detailed Description

Manages settings for each serial port

2.127.2 Member Function Documentation

2.127.2.1 UeiDaqAPI tUeiSerialPortMode UeiDaq::CUeiSerialPort::GetMode ()

Get the mode used by the serial port. Possible modes are RS-232, RS-485 half duplex and RS-485 full duplex

Returns:

the serial port mode .

See also:

SetMode (p. 392)

2.127.2.2 UeiDaqAPI void UeiDaq::CUeiSerialPort::SetMode (tUeiSerialPortMode *mode*)

Set the mode used by the serial port. Possible modes are RS-232, RS-485 half duplex and RS-485 full duplex

Parameters:

mode The new serial port mode

See also:

GetMode (p. 392)

2.127.2.3 UeiDaqAPI tUeiSerialPortSpeed UeiDaq::CUeiSerialPort::GetSpeed ()

Get the number of data bits transmitted per second.

Returns:

the serial port speed.

See also:

SetSpeed (p. 393)

2.127.2.4 UeiDaqAPI void UeiDaq::CUeiSerialPort::SetSpeed (tUeiSerialPortSpeed *bitsPerSecond*)

Set the number of data bits transmitted per second.

Parameters:

bitsPerSecond The serial port speed

See also:

GetSpeed (p. 392)

2.127.2.5 UeiDaqAPI uInt32 UeiDaq::CUeiSerialPort::GetCustomSpeed ()

Get the number of data bits transmitted per second. Call this method when using a non-standard port speed

Returns:

the serial port speed.

See also:

SetCustomSpeed (p. 393)

2.127.2.6 UeiDaqAPI void UeiDaq::CUeiSerialPort::SetCustomSpeed (uInt32 *bitsPerSecond*)

Set the number of data bits transmitted per second. Call this method when using a non-standard port speed

Parameters:

bitsPerSecond The serial port speed

See also:

GetCustomSpeed (p. 393)

2.127.2.7 UeiDaqAPI tUeiSerialPortDataBits UeiDaq::CUeiSerialPort::GetDataBits ()

Get the number of data bits that hold the data to be transferred.

Returns:

the number of data bits.

See also:

SetDataBits (p. 394)

2.127.2.8 UeiDaqAPI void UeiDaq::CUeiSerialPort::SetDataBits (tUeiSerialPortDataBits *dataBits*)

Set the number of data bits that hold the data to be transferred.

Parameters:

dataBits The number of data bits

See also:

GetDataBits (p. 393)

2.127.2.9 UeiDaqAPI tUeiSerialPortParity UeiDaq::CUeiSerialPort::GetParity ()

Get the parity used to detect transmission errors.

Returns:

the parity.

See also:

SetParity (p. 394)

2.127.2.10 UeiDaqAPI void UeiDaq::CUeiSerialPort::SetParity (tUeiSerialPortParity *parity*)

Set the parity used to detect transmission errors.

Parameters:

parity The serial port parity

See also:

GetParity (p. 394)

2.127.2.11 UeiDaqAPI tUeiSerialPortStopBits UeiDaq::CUeiSerialPort::GetStopBits ()

Get the number of stop bits that indicate the end of a data message.

Returns:

the number of stop bits.

See also:

SetStopBits (p. 395)

2.127.2.12 UeiDaqAPI void UeiDaq::CUeiSerialPort::SetStopBits (tUeiSerialPortStopBits *stopBits*)

Set the number of stop bits that indicate the end of a data message.

Parameters:

stopBits The number of stop bits

See also:

GetStopBits (p. 394)

2.127.2.13 UeiDaqAPI std::string UeiDaq::CUeiSerialPort::GetTermination (void)

Get the sequence of characters used to define the end of a line.

Returns:

The termination characters

See also:

SetTermination (p. 395)

2.127.2.14 UeiDaqAPI void UeiDaq::CUeiSerialPort::SetTermination (std::string *eol*)

Set the sequence of characters used to define the end of a line.

Parameters:

eol The termination characters

See also:

GetTermination (p. 395)

2.127.2.15 UeiDaqAPI bool UeiDaq::CUeiSerialPort::IsTxTerminationResistorEnabled ()

Determines whether the TX termination resistor is enabled

Returns:

The TX termination resistor state.

See also:

EnableTxTerminationResistor (p. 396)

2.127.2.16 UeiDaqAPI void UeiDaq::CUeiSerialPort::EnableTxTerminationResistor (bool *enabled*)

Enable or disable the TX termination resistor.

Parameters:

enabled The new TX termination resistor state

See also:

IsTxTerminationResistorEnabled (p. 395)

2.127.2.17 UeiDaqAPI bool UeiDaq::CUeiSerialPort::IsRxTerminationResistorEnabled ()

Determines whether the RX termination resistor is enabled

Returns:

The RX termination resistor state.

See also:

EnableRxTerminationResistor (p. 396)

2.127.2.18 UeiDaqAPI void UeiDaq::CUeiSerialPort::EnableRxTerminationResistor (bool *enabled*)

Enable or disable the RX termination resistor.

Parameters:

enabled The new RX termination resistor state

See also:

IsRxTerminationResistorEnabled (p. 396)

2.127.2.19 UeiDaqAPI bool UeiDaq::CUeiSerialPort::IsErrorReportingEnabled ()

Determines whether the framing errors or parity errors will be reported

Returns:

The error reporting state.

See also:

EnableErrorReporting (p. 397)

2.127.2.20 UeiDaqAPI void UeiDaq::CUeiSerialPort::EnableErrorReporting (bool *enabled*)

Enable or disable reproting of framing and parity errors.

Parameters:

enabled The new error reporting state

See also:

IsErrorReportingEnabled (p. 396)

2.127.2.21 UeiDaqAPI tUeiSerialPortFlowControl UeiDaq::CUeiSerialPort::GetFlowControl ()

Get flow control setting

Returns:

The current flow control setting

See also:

SetFlowControl (p. 397)

2.127.2.22 UeiDaqAPI void UeiDaq::CUeiSerialPort::SetFlowControl (tUeiSerialPortFlowControl *flowControl*)

Set flow control setting

Parameters:

flowControl The new flow control setting

See also:

GetFlowControl (p. 397)

2.127.2.23 UeiDaqAPI bool UeiDaq::CUeiSerialPort::IsHDEchoSuppressionEnabled ()

Determines whether the half duplex echo suppression is enabled

Returns:

The half-duplex echo suppression state.

See also:

EnableHDEchoSuppression (p. 398)

2.127.2.24 UeiDaqAPI void UeiDaq::CUeiSerialPort::EnableHDEchoSuppression (bool *enabled*)

Enable or disable the half-duplex echo suppression.

Parameters:

enabled The new half-duplex echo suppression state

See also:

IsHDEchoSuppressionEnabled (p. 397)

2.127.2.25 UeiDaqAPI bool UeiDaq::CUeiSerialPort::IsTxAutoDisableEnabled ()

Determines whether transmitter should automatically disable itself whenever the TX FIFO is empty (only used in RS-485 full duplex mode) This allows multiple RS-485 devices to share the same bus

Returns:

The TX auto-disable state

See also:

EnableTxAutoDisable (p. 398)

2.127.2.26 UeiDaqAPI void UeiDaq::CUeiSerialPort::EnableTxAutoDisable (bool *enabled*)

Specifies whether transmitter should automatically disable itself whenever the TX FIFO is empty (only used in RS-485 full duplex mode) This allows multiple RS-485 devices to share the same bus

Parameters:

enabled The new TX auto-disable state

See also:

IsTxAutoDisableEnabled (p. 398)

2.127.2.27 UeiDaqAPI bool UeiDaq::CUeiSerialPort::IsOnTheFlyParityBitEnabled ()

Determines whether the parity bit can be specified on the fly as part of the data stream (using uInt16 data instead of uInt8)

Returns:

The parity bit state.

See also:

EnableOnTheFlyParityBit (p. 399)

2.127.2.28 UeiDaqAPI void UeiDaq::CUeiSerialPort::EnableOnTheFlyParityBit (bool *enabled*)

Enable or disable on the fly parity bit update as part of the data stream (using uInt16 data instead of uInt8)

Parameters:

enabled The new on the fly parity bit state

See also:

IsOnTheFlyParityBitEnabled (p. 398)

2.127.2.29 UeiDaqAPI bool UeiDaq::CUeiSerialPort::IsTimeoutUponReceiveEnabled ()

Determines whether timeout error is thrown when not enough bytes are available for retrieval.

Returns:

enabled The new timeout upon read state

See also:

EnableTimeoutUponReceive (p. 399)

2.127.2.30 UeiDaqAPI void UeiDaq::CUeiSerialPort::EnableTimeoutUponReceive (bool *enabled*)

Enable or disable timeout error when not enough bytes are available for retrieval.

Parameters:

enabled The new timeout upon read state

See also:

IsTimeoutUponReceiveEnabled (p. 399)

2.127.2.31 UeiDaqAPI uInt32 UeiDaq::CUeiSerialPort::GetCharDelay ()

Get the delay between transmitted characters (useful when communicating with slow devices)

Returns:

The current character delay in microseconds

2.127.2.32 UeiDaqAPI void UeiDaq::CUeiSerialPort::SetCharDelay (uInt32 *charDelayUs*)

Configures a delay between transmitted characters (useful when communicating with slow devices)

Parameters:

charDelayUs The new character delay in microseconds

2.127.2.33 UeiDaqAPI tUeiSerialPortMinorFrameMode UeiDaq::CUeiSerialPort::GetMinorFrameMode ()

Get the mode used to group characters in minor frames

Returns:

The current frame mode

2.127.2.34 UeiDaqAPI void UeiDaq::CUeiSerialPort::SetMinorFrameMode (tUeiSerialPortMinorFrameMode *mode*)

Set the mode used to group characters in minor frames

Parameters:

mode The new frame mode

2.127.2.35 UeiDaqAPI uInt32 UeiDaq::CUeiSerialPort::GetMinorFrameLength ()

Get the minor frame length. This is only meaningful when frame mode is set to "fixed length"

Returns:

The current frame length

2.127.2.36 UeiDaqAPI void UeiDaq::CUeiSerialPort::SetMinorFrameLength (uInt32 *length*)

Set the minor frame length (number of characters per frame) This is only meaningful when frame mode is set to "fixed length"

Parameters:

length The new frame length

2.127.2.37 UeiDaqAPI uInt32 UeiDaq::CUeiSerialPort::GetMinorFrameDelay ()

Get the delay between transmitted minor frames

Returns:

The current minor frame delay in microseconds

2.127.2.38 UeiDaqAPI void UeiDaq::CUeiSerialPort::SetMinorFrameDelay (uInt32 *frameDelayUs*)

Set the delay between transmitted minor frames

Parameters:

frameDelayUs The new minor frame delay in microseconds

2.127.2.39 UeiDaqAPI uInt32 UeiDaq::CUeiSerialPort::GetMajorFramePeriod ()

Major framing pre-fills the FIFO with a sequence of characters (optionally including char delay and minor frames) and periodically transmits the FIFO content. It isn't possible to update the major frame data once it is started. Get the period used to re-transmit major frames

Returns:

The current major frame period in microseconds

2.127.2.40 UeiDaqAPI void UeiDaq::CUeiSerialPort::SetMajorFramePeriod (uInt32 *framePeriodUs*)

Major framing pre-fills the FIFO with a sequence of characters (optionally including char delay and minor frames) and periodically transmits the FIFO content. It isn't possible to update the major frame data once it is started. Get the period used to re-transmit major frames Set the period at which major frames are transmitted

Parameters:

framePeriodUs The new major frame period in microseconds

2.127.2.41 UeiDaqAPI f64 UeiDaq::CUeiSerialPort::GetTimestampResolution ()

Get the timestamp resolution in seconds.

Returns:

the timestamp resolution.

See also:

SetTimestampResolution (p. 401)

2.127.2.42 UeiDaqAPI void UeiDaq::CUeiSerialPort::SetTimestampResolution (f64 *resolution*)

Set the timestamp resolution in seconds.

Parameters:

resolution the new resolution.

See also:

GetTimestampResolution (p. 401)

2.127.2.43 UeiDaqAPI bool UeiDaq::CUeiSerialPort::IsLoopbackEnabled ()

Determines whether the loopback mode is enabled when enabled the port receives what it just transmitted

Returns:

The state.

See also:

EnableLoopback (p. 402)

2.127.2.44 UeiDaqAPI void UeiDaq::CUeiSerialPort::EnableLoopback (bool *enabled*)

Enable or disable the half-duplex echo suppression. when enabled the port receives what it just transmitted

Parameters:

enabled The new loopback state

See also:

IsLoopbackEnabled (p. 402)

2.127.2.45 UeiDaqAPI bool UeiDaq::CUeiSerialPort::IsBreakEnabled ()

Determines whether break detection is enabled when enabled serial break is used as a condition to determine end of serial messages

Returns:

The state.

See also:

EnableBreak (p. 402)

2.127.2.46 UeiDaqAPI void UeiDaq::CUeiSerialPort::EnableBreak (bool *enabled*)

Enable or disable the break detection. when enabled serial break is used as a condition to determine end of serial messages

Parameters:

enabled The new break detection state

See also:

IsBreakEnabled (p. 402)

2.127.2.47 UeiDaqAPI char UeiDaq::CUeiSerialPort::GetReceiveBreakCharacter ()

Get the special character inserted in the byte stream upon reception of a serial break

Returns:

The break cahracter.

See also:

SetReceiveBreakCharacter (p. 403)

2.127.2.48 UeiDaqAPI void UeiDaq::CUeiSerialPort::SetReceiveBreakCharacter (char *breakChar*)

Set the special character inserted in the byte stream upon reception of a serial break

Parameters:

breakChar The new break character

See also:

GetReceiveBreakCharacter (p. 403)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.128 UeiDaq::CUeiSerialReader Class Reference

Serial Reader class.

`#include <UeiMessaging.h>`

Public Member Functions

- UeiDaqAPI **CUeiSerialReader** (**CUeiDataStream** *pDataStream, **Int32** port=0)
  Constructor.
- virtual UeiDaqAPI ~**CUeiSerialReader** ()
  Destructor.
- UeiDaqAPI void **Read** (**Int32** bufferSize, **Int8** *pBuffer, **Int32** *numBytesRead)
  Read data from the serial port.
- UeiDaqAPI void **ReadTimestamped** (**Int32** bufferSize, **Int8** *pBuffer, **Int32** *numBytesRead)
  Read data from the serial port, returning the timestamp as the first sizeof(uInt32) bytes in the pBuffer.
- UeiDaqAPI void **ReadAsync** (**Int32** bufferSize, **Int8** *pBuffer)
  Read data from the serial port asynchronously.
- UeiDaqAPI void **AddEventListener** (**IUeiEventListener** *pListener)
  Add an asynchronous listener.

2.128.1 Detailed Description

Class that handles reading data messages of a stream coming from a serial interface

2.128.2 Constructor & Destructor Documentation

2.128.2.1 UeiDaqAPI UeiDaq::CUeiSerialReader::CUeiSerialReader (CUeiDataStream * *pDataStream*, Int32 *port* = 0)

Parameters:

pDataStream represents the source of data to read

port The serial port to read data from

2.128.3 Member Function Documentation

2.128.3.1 UeiDaqAPI void UeiDaq::CUeiSerialReader::Read (Int32 *bufferSize*, Int8 * *pBuffer*, Int32 * *numBytesRead*)

Read a data message from the input stream

Parameters:

bufferSize the size of the destination buffer

pBuffer destination buffer

numBytesRead the actual number of bytes read from the serial port

2.128.3.2 UeiDaqAPI void UeiDaq::CUeiSerialReader::ReadTimestamped (Int32 *bufferSize*, Int8 * *pBuffer*, Int32 * *numBytesRead*)

Read a data message from the input stream placing the timestamp in the first sizeof(uInt32) bytes of pBuffer. An additional sizeof(uInt32) bytes should be added to bufferSize and the size of pBuffer to accomodate for the timestamp.

Parameters:

bufferSize the size of the destination buffer

pBuffer destination buffer

numBytesRead the actual number of bytes read from the serial port

2.128.3.3 UeiDaqAPI void UeiDaq::CUeiSerialReader::ReadAsync (Int32 *bufferSize*, Int8 * *pBuffer*)

Read a data message from the input stream

Parameters:

bufferSize the size of the destination buffer

pBuffer destination buffer

2.128.3.4 UeiDaqAPI void UeiDaq::CUeiSerialReader::AddEventListener (IUeiEventListener * *pListener*)

Subscribe a listener to receive asynchronous events

Parameters:

pListener pointer to a class that implements **IUeiEventListener** (p. 549) interface

The documentation for this class was generated from the following file:

- UeiMessaging.h

2.129 UeiDaq::CUeiSerialWriter Class Reference

Serial Writer class.

`#include <UeiMessaging.h>`

Public Member Functions

- UeiDaqAPI **CUeiSerialWriter** (**CUeiDataStream** *pDataStream, **Int32** port=0)
  Constructor.
- virtual UeiDaqAPI ~**CUeiSerialWriter** ()
  Destructor.
- UeiDaqAPI void **Write** (**Int32** bufferSize, **Int8** *pBuffer, **Int32** *numBytesWritten)
  Write 8-bit data to the serial port.
- UeiDaqAPI void **Write** (**Int32** bufferSize, **Int16** *pBuffer, **Int32** *numBytesWritten)
  Write 16-bit data to the serial port.
- UeiDaqAPI void **WriteAsync** (**Int32** bufferSize, **Int8** *pBuffer)
  Write data to the serial port asynchronously.
- UeiDaqAPI void **SendBreak** (**uInt32** durationMs)
  Send serial break.
- UeiDaqAPI void **AddEventListener** (**IUeiEventListener** *pListener)
  Add an asynchronous listener.

2.129.1 Detailed Description

Class that handles writing data messages to a stream going through a serial interface

2.129.2 Constructor & Destructor Documentation

2.129.2.1 UeiDaqAPI UeiDaq::CUeiSerialWriter::CUeiSerialWriter (CUeiDataStream * *pDataStream*, Int32 *port* = 0)

Parameters:

pDataStream represents the destination where to write data
port the serial port to write data to

2.129.3 Member Function Documentation

2.129.3.1 UeiDaqAPI void UeiDaq::CUeiSerialWriter::Write (Int32 *bufferSize*, Int8 * *pBuffer*, Int32 * *numBytesWritten*)

Write 8-bit data message to the serial port

Parameters:

bufferSize the size of the source buffer

pBuffer source buffer

numBytesWritten the actual number of bytes written to the serial port

2.129.3.2 UeiDaqAPI void UeiDaq::CUeiSerialWriter::Write (Int32 *bufferSize*, Int16 * *pBuffer*, Int32 * *numBytesWritten*)

Write 16-bit data message to the serial port, useful when setting parity bit on the fly

Parameters:

bufferSize the size of the source buffer

pBuffer source buffer

numBytesWritten the actual number of bytes written to the serial port

2.129.3.3 UeiDaqAPI void UeiDaq::CUeiSerialWriter::WriteAsync (Int32 *bufferSize*, Int8 * *pBuffer*)

Write a data message to the serial port

Parameters:

bufferSize the size of the source buffer

pBuffer source buffer

2.129.3.4 UeiDaqAPI void UeiDaq::CUeiSerialWriter::SendBreak (uInt32 *durationMs*)

Write a break to the serial port

Parameters:

durationMs break duration in ms

2.129.3.5 UeiDaqAPI void UeiDaq::CUeiSerialWriter::AddEventListener (IUeiEventListener * *pListener*)

Subscribe a listener to receive asynchronous events

Parameters:

pListener pointer to a class that implements **IUeiEventListener** (p. 549) interface

The documentation for this class was generated from the following file:

- UeiMessaging.h

2.130 UeiDaq::CUeiSession Class Reference

Session Class.

`#include <UeiSession.h>`

Inherits **UeiDaq::CUeiObject**.

Public Member Functions

- UeiDaqAPI **CUeiSession** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiSession** ()
  Destructor.
- UeiDaqAPI **CUeiDiagnosticChannel** * **CreateDiagnosticChannel** (std::string resource)
  Create Diagnostic input channel list.
- UeiDaqAPI **CUeiAIChannel** * **CreateAIChannel** (std::string resource, **f64** min, **f64** max, **tUeiAIChannelInputMode** mode)
  Create Analog voltage input channel list.
- UeiDaqAPI **CUeiAICurrentChannel** * **CreateAICurrentChannel** (std::string resource, **f64** min, **f64** max, **tUeiFeatureEnable** enableCB, **tUeiAIChannelInputMode** mode)
  Create Analog current input channel list.
- UeiDaqAPI **CUeiAOChannel** * **CreateAOChannel** (std::string resource, **f64** min, **f64** max)
  Create Analog output voltage channel list.
- UeiDaqAPI **CUeiAOCurrentChannel** * **CreateAOCurrentChannel** (std::string resource, **f64** min, **f64** max)
  Create Analog output current channel list.
- UeiDaqAPI **CUeiAOWaveformChannel** * **CreateAOWaveformChannel** (std::string resource, **tUeiAOWaveformClockSource** dacClockSource, **tUeiAOWaveformOffsetClockSource** offsetDACClockSource, **tUeiAOWaveformClockSync** clockSync)
  Create Analog output waveform channel list.
- UeiDaqAPI **CUeiAOProtectedChannel** * **CreateAOProtectedChannel** (std::string resource, **tUeiAODACMode** mode, double measurementRate, bool autoRetry, double retryRate)
  Create Protected analog output voltage channel list.
- UeiDaqAPI **CUeiAOProtectedCurrentChannel** * **CreateAOProtectedCurrentChannel** (std::string resource, **tUeiAODACMode** mode, double measurementRate, bool autoRetry, double retryRate)
  Create Protected analog output current channel list.
- UeiDaqAPI **CUeiSimulatedTCChannel** * **CreateSimulatedTCChannel** (std::string resource, **tUeiThermocoupleType** tcType, **tUeiTemperatureScale** tempScale, bool enableCjc)

Create Simulated Thermocouple output channel list.

- UeiDaqAPI **CUeiSimulatedRTDChannel** * **CreateSimulatedRTDChannel** (std::string resource, **tUeiRTDType** rtdType, double rtdNominalResistance, **tUeiTemperatureScale** tempScale)

  Create simulated RTD output channel list.

- UeiDaqAPI **CUeiDIChannel** * **CreateDIChannel** (std::string resource)

  Create Digital input channel list.

- UeiDaqAPI **CUeiDIIndustrialChannel** * **CreateDIIndustrialChannel** (std::string resource, double lowThreshold, double highThreshold, double minPulseWidth)

  Create industrial digital input channel list.

- UeiDaqAPI **CUeiDOChannel** * **CreateDOChannel** (std::string resource)

  Create Digital output channel list.

- UeiDaqAPI **CUeiDOIndustrialChannel** * **CreateDOIndustrialChannel** (std::string resource, **tUeiDOPWMMode** pwmMode, **uInt32** pwmLengthUs, **uInt32** pwmPeriodUs, double pwmDutyCycle)

  Create Industrial digital output channel list.

- UeiDaqAPI **CUeiDOProtectedChannel** * **CreateDOProtectedChannel** (std::string resource, double underCurrentLimit, double overCurrentLimit, double currentSampleRate, bool autoRetry, double retryRate)

  Create Protected digital output channel list.

- UeiDaqAPI **CUeiCIChannel** * **CreateCIChannel** (std::string resource, **tUeiCounterSource** source, **tUeiCounterMode** mode, **tUeiCounterGate** gate, **Int32** divider, **Int32** inverted)

  Create Counter input channel list.

- UeiDaqAPI **CUeiQuadratureEncoderChannel** * **CreateQuadratureEncoderChannel** (std::string resource, **uInt32** initialPosition, **tUeiQuadratureDecodingType** decodingType, bool enableZeroIndexing, **tUeiQuadratureZeroIndexPhase** zeroIndexPhase)

  Create quadrature encoder input channel list.

- UeiDaqAPI **CUeiVRChannel** * **CreateVRChannel** (std::string resource, **tUeiVRMode** mode)

  Create variable reluctance input channel list.

- UeiDaqAPI **CUeiSSIMasterPort** * **CreateSSIMasterPort** (std::string resource, unsigned int bps, unsigned int wordSize, bool enableClock, bool enableTermination, double pauseTime, double transferTimeout, double bitUpdateTime)

  Create SSI master port list.

- UeiDaqAPI **CUeiSSISlavePort** * **CreateSSISlavePort** (std::string resource, unsigned int bps, unsigned int wordSize, bool enableTransmit, bool enableTermination, double pauseTime, double transferTimeout, double bitUpdateTime)

  Create SSI slave port list.

- UeiDaqAPI **CUeiCOChannel** * **CreateCOChannel** (std::string resource, **tUeiCounterSource** source, **tUeiCounterMode** mode, **tUeiCounterGate** gate, **uInt32** tick1, **uInt32** tick2, **Int32** divider, **Int32** inverted)

  Create Counter output channel list.

- UeiDaqAPI **CUeiTCChannel** * **CreateTCChannel** (std::string resource, **f64** min, **f64** max, **tUeiThermocoupleType** tcType, **tUeiTemperatureScale** tempScale, **tUeiColdJunctionCompensationType** cjcType, **f64** cjcConstant, std::string cjcResource, **tUeiAIChannelInputMode** mode)

  Create Thermocouple input channel list.

- UeiDaqAPI **CUeiRTDChannel** * **CreateRTDChannel** (std::string resource, **f64** min, **f64** max, **tUeiWiringScheme** wiring, double twoWiresLeadResistance, **tUeiRTDType** rtdType, double rtdNominalResistance, **tUeiTemperatureScale** tempScale, **tUeiAIChannelInputMode** mode)

  Create RTD input channel list.

- UeiDaqAPI **CUeiResistanceChannel** * **CreateResistanceChannel** (std::string resource, **f64** min, **f64** max, **tUeiWiringScheme** wiring, double twoWiresLeadResistance, **tUeiAIChannelInputMode** mode)

  Create resistance input channel list.

- UeiDaqAPI **CUeiAIVExChannel** * **CreateAIVExChannel** (std::string resource, **f64** min, **f64** max, **tUeiSensorBridgeType** sensorBridgeType, double excitationVoltage, bool bUseExcitationForScaling, **tUeiAIChannelInputMode** mode)

  Create Analog input with excitation channel list.

- UeiDaqAPI **CUeiAccelChannel** * **CreateAccelChannel** (std::string resource, **f64** min, **f64** max, double sensorSensitivity, double excitationCurrent, **tUeiCoupling** coupling, bool enableLowPassFilter)

  Create Accelerometer channel list.

- UeiDaqAPI **CUeiDMMChannel** * **CreateDMMChannel** (std::string resource, **f64** range, **tUeiDMMMeasurementMode** measurementMode)

  Create digital multi-meter channel.

- UeiDaqAPI **CUeiLVDTChannel** * **CreateLVDTChannel** (std::string resource, **f64** min, **f64** max, double sensorSensitivity, **tUeiLVDTWiringScheme** wiringScheme, double excitationVoltage, double excitationFrequency, bool externalExcitation)

  Create LVDT/RVDT analog input channel list.

- UeiDaqAPI **CUeiSimulatedLVDTChannel** * **CreateSimulatedLVDTChannel** (std::string resource, double sensorSensitivity, **tUeiLVDTWiringScheme** wiringScheme, double excitationVoltage, double excitationFrequency)

  Create simulated LVDT/RVDT analog output channel list.

- UeiDaqAPI **CUeiSynchroResolverChannel** * **CreateSynchroResolverChannel** (std::string resource, **tUeiSynchroResolverMode** mode, double excitationVoltage, double excitationFrequency, bool externalExcitation)

  Create Synchro/Resolver analog input channel list.

- UeiDaqAPI **CUeiSimulatedSynchroResolverChannel** * **CreateSimulatedSynchroResolverChannel** (std::string resource, **tUeiSynchroResolverMode** mode, double excitationVoltage, double excitationFrequency, bool externalExcitation)

  Create simulated Synchro/Resolver analog output channel list.

- UeiDaqAPI **CUeiSerialPort** * **CreateSerialPort** (std::string resource, **tUeiSerialPortMode** mode, **tUeiSerialPortSpeed** bitsPerSecond, **tUeiSerialPortDataBits** dataBits, **tUeiSerialPortParity** parity, **tUeiSerialPortStopBits** stopBits, std::string termination)

  Create Serial port.

- UeiDaqAPI **CUeiHDLCPort** * **CreateHDLCPort** (std::string resource, **tUeiHDLCPortPhysical** physInterface, **uInt32** bitsPerSecond, **tUeiHDLCPortEncoding** encoding, **tUeiHDLCPortCRCMode** crcMode, **tUeiHDLCPortClockSource** txClockSource, **tUeiHDLCPortClockSource** rxClockSource)

  Create HDLC port.

- UeiDaqAPI **CUeiCANPort** * **CreateCANPort** (std::string resource, **tUeiCANPortSpeed** bitsPerSecond, **tUeiCANFrameFormat** frameFormat, **tUeiCANPortMode** mode, **uInt32** acceptanceMask, **uInt32** acceptanceCode)

  Create CAN port.

- UeiDaqAPI **CUeiARINCInputPort** * **CreateARINCInputPort** (std::string resource, **tUeiARINCPortSpeed** bitsPerSecond, **tUeiARINCPortParity** parity, bool enableSDIFilter, **uInt32** SDIMask)

  Create ARINC input port.

- UeiDaqAPI **CUeiARINCOutputPort** * **CreateARINCOutputPort** (std::string resource, **tUeiARINCPortSpeed** bitsPerSecond, **tUeiARINCPortParity** parity)

  Create ARINC output port.

- UeiDaqAPI **CUeiMIL1553Port** * **CreateMIL1553Port** (std::string resource, **tUeiMIL1553PortCoupling** coupling, **tUeiMIL1553PortOpMode** portMode)

  Create MIL-1553 port.

- UeiDaqAPI **CUeiIRIGTimeKeeperChannel** * **CreateIRIGTimeKeeperChannel** (std::string resource, **tUeiIRIGTimeKeeper1PPSSource** source, bool autoFollow)

  Create IRIG time keeper channel.

- UeiDaqAPI **CUeiIRIGInputChannel** * **CreateIRIGInputChannel** (std::string resource, **tUeiIRIGDecoderInputType** inputType, **tUeiIRIGTimeCodeFormat** timeCodeFormat)

  Create IRIG input channel.

- UeiDaqAPI **CUeiIRIGOutputChannel** * **CreateIRIGOutputChannel** (std::string resource, **tUeiIRIGTimeCodeFormat** timeCodeFormat)

  Create IRIG output channel.

- UeiDaqAPI **CUeiIRIGDOTTLChannel** * **CreateIRIGDOTTLChannel** (std::string resource, **tUeiIRIGDOTTLSource** source0, **tUeiIRIGDOTTLSource** source1, **tUeiIRIGDOTTLSource** source2, **tUeiIRIGDOTTLSource** source3)

  Create IRIG DO TTL channel.

- UeiDaqAPI **CUeiSync1PPSPort** * **CreateSync1PPSPort** (std::string resource, **tUeiSync1PPSMode** mode, **tUeiSync1PPSSource** source, double EMOutputRate)

  Create 1PPS synchronization port.

- UeiDaqAPI **CUeiCSDBPort** * **CreateCSDBPort** (std::string resource, int bps, int parity, int blockSize, int numberOfMessages, int interByteDelayUs, int interBlockDelayUs, int framePeriodUs)

  Create CSDB port.

- UeiDaqAPI **CUeiMuxPort** * **CreateMuxPort** (std::string resource, bool breakBeforeMake)

  Create Mux port.

- UeiDaqAPI **CUeiI2CMasterPort** * **CreateI2CMasterPort** (std::string resource, **tUeiI2CPortSpeed** bitRate, **tUeiI2CTTLLevel** ttlLevel, bool enableSecureShell)

  Create I2C master port.

- UeiDaqAPI **CUeiI2CSlavePort** * **CreateI2CSlavePort** (std::string resource, **tUeiI2CTTLLevel** ttlLevel, **tUeiI2CSlaveAddressWidth** addressWidth, int address)

  Create I2C slave port.

- UeiDaqAPI void **CreateInfoSession** (std::string resource)

  Create info session.

- UeiDaqAPI void **ConfigureTimingForSimpleIO** ()

  Configure the timing object for simple IO.

- UeiDaqAPI void **ConfigureTimingForBufferedIO** (int samplePerChannel, **tUeiTimingClockSource** clkSource, double rate, **tUeiDigitalEdge** edge, **tUeiTimingDuration** duration)

  Configure the timing object for buffered mode.

- UeiDaqAPI void **ConfigureTimingForDataMappingIO** (**tUeiTimingClockSource** clkSource, double rate)

  Configure the timing object for data map mode.

- UeiDaqAPI void **ConfigureTimingForEdgeDetection** (**tUeiDigitalEdge** edge)

  Configure the timing object for digital line state change detection.

- UeiDaqAPI void **ConfigureTimingForAsynchronousIO** (int watermark, int periodUs, int noActivityTimeoutUs, int maxDataSize)

  Configure the timing object for asynchronous FIFO based I/O.

- UeiDaqAPI void **ConfigureTimingForTimeSequencing** (int samplePerChannel, **tUeiTimingDuration** duration)

  Configure the timing object for time sequencing.

- UeiDaqAPI void **ConfigureTimingForMessagingIO** (int bufferSize, double refreshRate)

  Configure the timing object for messaging IO.

- UeiDaqAPI void **ConfigureTimingForVMapIO** (double rate)

  Configure the timing object for variable map mode.

- UeiDaqAPI void **ConfigureStartDigitalTrigger** (**tUeiTriggerSource** source, **tUeiDigitalEdge** edge)

  Configure the digital trigger condition that will start the session.

- UeiDaqAPI void **ConfigureStopDigitalTrigger** (**tUeiTriggerSource** source, **tUeiDigitalEdge** edge)

  Configure the digital trigger condition that will stop the session.

- UeiDaqAPI void **ConfigureAnalogSoftwareTrigger** (**tUeiTriggerAction** action, **tUeiTriggerCondition** condition, **Int32** triggerChannel, **f64** level, **f64** hysteresis, **Int32** preTriggerScans)

  Configure the analog software trigger.

- UeiDaqAPI void **ConfigureSignalTrigger** (**tUeiTriggerAction** action, std::string signal)

  Configure the trigger signal that will start or stop the session.

- UeiDaqAPI void **AddEventListener** (**IUeiEventListener** *pListener)

  Add an event listener to the session.

- UeiDaqAPI void **Start** ()

  Start the session.

- UeiDaqAPI bool **IsRunning** ()

  Check session status.

- UeiDaqAPI bool **WaitUntilDone** (int timeout)

  Wait for the specified time until the session is stopped.

- UeiDaqAPI void **Stop** ()

  Stop the session.

- UeiDaqAPI int **Pause** (int port)

  Pause one port in the session.

- UeiDaqAPI void **Resume** (int port)

  Pause one port in the session.

- UeiDaqAPI void **CleanUp** ()

  Clean-up the session object.

- UeiDaqAPI void **EnableAutoReStart** (bool enable)

  Enable/Disable session auto-restart.

- UeiDaqAPI **CUeiDevice** * **GetDevice** ()

  Get a pointer to the device object.

- UeiDaqAPI **CUeiTiming** * **GetTiming** ()

  Get a pointer to the timing object.

- UeiDaqAPI **CUeiDataStream** * **GetDataStream** ()
  Get a pointer to the stream object.
- UeiDaqAPI **CUeiTrigger** * **GetStartTrigger** ()
  Get a pointer to the start trigger object.
- UeiDaqAPI **CUeiTrigger** * **GetStopTrigger** ()
  Get a pointer to the stop trigger object.
- UeiDaqAPI int **GetNumberOfChannels** ()
  Get the number of channels.
- UeiDaqAPI **CUeiChannel** * **GetChannel** (int index)
  Get a channel object by position index in channel list.
- UeiDaqAPI **CUeiChannel** * **GetChannelById** (int id)
  Get a channel object by physical channel id.
- UeiDaqAPI void **SetCustomProperty** (std::string property, int valueSize, void *value)
  Set a custom property.
- UeiDaqAPI void **SetCustomProperty** (std::string property, int value)
  Set an integer custom property.
- UeiDaqAPI void **SetCustomProperty** (std::string property, double value)
  Set a double custom property.
- UeiDaqAPI void **SetCustomProperty** (std::string property, int arraySize, int *valueArray)
  Set an integer array custom property.
- UeiDaqAPI void **SetCustomProperty** (std::string property, int arraySize, double *value-Array)
  Set a double array custom property.
- UeiDaqAPI void **GetCustomProperty** (std::string property, int valueSize, void *value)
  Get a custom property.
- UeiDaqAPI void **GetCustomProperty** (std::string property, int *value)
  Get an integer custom property.
- UeiDaqAPI void **GetCustomProperty** (std::string property, double *value)
  Get a double custom property.
- UeiDaqAPI void **GetCustomProperty** (std::string property, int arraySize, int *valueArray)
  Get an integer array custom property.
- UeiDaqAPI void **GetCustomProperty** (std::string property, int arraySize, double *value-Array)
  Get a double array custom property.

- UeiDaqAPI **tUeiSessionType GetType** ()

  Get the session type.

- UeiDaqAPI int **GetSubsystem** ()

  Get the session susbsytem index.

- UeiDaqAPI void **LoadFromXml** (std::string xmlSettings)

  Load Session settings from an XML stream.

- UeiDaqAPI void **LoadFromFile** (std::string sessionFile)

  Load Session settings from a file.

2.130.1 Detailed Description

The session class is used to store the DAQ device settings and control its operation

```
#include <iostream>
#include "UeiDaq.h"

using namespace UeiDaq;

int main(int argc, char* argv[])
{
   CUeiSession mySs;
   double data[800];

   try
   {
      // Create 8 analog input channels on a powerdaq board
      // From now on the session is AI only
      mySs.CreateAIChannel("simu://Dev0/ai0:7", -10.0, 10.0, UeiAIChannelInputModeDifferential);

      // By default the timing object is configured for simple I/O so
      // no need to do anything here.
      mySs.ConfigureTimingForSimpleIO();

      // Create a reader object to read data synchronously.
      CUeiAnalogScaledReader reader(mySs.GetDataStream());

      mySs.Start();

      // Read 100 scans
      for(int i=0; i<100; i++)
      {
         reader.ReadSingleScan(&data[i*mySs.GetNumberOfChannels()]);
         std::cout << "Ch0 = " << data[i*mySs.GetNumberOfChannels()] << std::endl;
      }

      mySs.Stop();
   }
   catch(CUeiException e)
   {
      std::cout << "Error: " << e.GetErrorMessage() << std::endl;
   }

   return 0;
}
```

Examples:

AnalogInBuffered.cpp, **AnalogInBufferedAsync.cpp**, **AnalogInOneShot_Trig_Ex.cpp**, **AnalogInSingle.cpp**, **AnalogOutBuffered.cpp**, **AnalogOutBufferedAsync.cpp**, **AnalogOutSingle.cpp**, **BufferedEventCounting.cpp**, **DigitalInEvent.cpp**, **EventCount.cpp**, **GeneratePulseTrain.cpp**, and **MeasureFrequency.cpp**.

2.130.2 Member Function Documentation

2.130.2.1 UeiDaqAPI CUeiDiagnosticChannel* UeiDaq::CUeiSession::CreateDiagnosticChannel (std::string *resource*)

Create and add one or more channel to the diagnostic channel list associated with the session.

Parameters:

resource The device and channel(s) to add to the list.

Returns:

A pointer to the first channel created.

2.130.2.2 UeiDaqAPI CUeiAIChannel* UeiDaq::CUeiSession::CreateAIChannel (std::string *resource*, f64 *min*, f64 *max*, tUeiAIChannelInputMode *mode*)

Create and add one or more channel to the AI channel list associated with the session.

Example:

```
CUeiSession mySession;

mySession.CreateAIChannel("pwrdaq://Dev1/Ai0", -10.0, 10.0, UeiAIChannelInputModeDifferential );
```

Parameters:

resource The device and channel(s) to add to the list.

min The minimum value you expect to measure.

max The maximum value you expect to measure

mode The input mode of the Analog input(s)

Returns:

A pointer to the first channel created.

See also:

CreateAOChannel (p. 417), **CreateDIChannel** (p. 420), **CreateDOChannel** (p. 421), **CreateCIChannel** (p. 423), **CreateCOChannel** (p. 425)

Examples:

AnalogInBuffered.cpp, **AnalogInBufferedAsync.cpp**, **AnalogInOneShot_Trig_Ex.cpp**, and **AnalogInSingle.cpp**.

2.130.2.3 UeiDaqAPI CUeiAICurrentChannel* UeiDaq::CUeiSession::CreateAICurrentChannel (std::string *resource*, f64 *min*, f64 *max*, tUeiFeatureEnable *enableCB*, tUeiAIChannelInputMode *mode*)

Create and add one or more channel to the AI channel list associated with the session.

Example:

```
CUeiSession mySession;

mySession.CreateAICurrentChannel("pdna://192.168.100.2/Dev1/Ai0", -10.0, 10.0, UeiFeatureDisabled, UeiAIChannelInput
```

Parameters:

resource The device and channel(s) to add to the list.

min The minimum value you expect to measure.

max The maximum value you expect to measure

enableCB The circuit breaker configuration

mode The input mode of the Analog input(s)

Returns:

A pointer to the first channel created.

See also:

CreateAIChannel (p. 416)

2.130.2.4 UeiDaqAPI CUeiAOChannel* UeiDaq::CUeiSession::CreateAOChannel (std::string *resource*, f64 *min*, f64 *max*)

Create and add one or more voltage channel to the AO channel list associated with the session. Voltage channels are only available on AO devices capable of outputting voltage signals

Parameters:

resource The device and channel(s) to add to the list.

min The minimum value you expect to generate.

max The maximum value you expect to generate

Returns:

A pointer to the first channel created.

See also:

CreateAIChannel (p. 416), **CreateDIChannel** (p. 420), **CreateDOChannel** (p. 421), **CreateCIChannel** (p. 423), **CreateCOChannel** (p. 425)

Examples:

AnalogOutBuffered.cpp, **AnalogOutBufferedAsync.cpp**, and **AnalogOutSingle.cpp**.

2.130.2.5 UeiDaqAPI CUeiAOCurrentChannel* UeiDaq::CUeiSession::CreateAOCurrentChannel (std::string *resource*, f64 *min*, f64 *max*)

Create and add one or more current channel to the AO current channel list associated with the session. Current channels are only available on AO devices capable of outputing current

Parameters:

resource The device and channel(s) to add to the list.

min The minimum value you expect to generate.

max The maximum value you expect to generate

Returns:

A pointer to the first channel created.

See also:

CreateAIChannel (p. 416), **CreateDIChannel** (p. 420), **CreateDOChannel** (p. 421), **CreateCIChannel** (p. 423), **CreateCOChannel** (p. 425)

2.130.2.6 UeiDaqAPI CUeiAOWaveformChannel* UeiDaq::CUeiSession::CreateAOWaveformChannel (std::string *resource*, tUeiAOWaveformClockSource *dacClockSource*, tUeiAOWaveformOffsetClockSource *offsetDACClockSource*, tUeiAOWaveformClockSync *clockSync*)

Create and add one or more waveform channel to the channel list associated with the session.

Parameters:

resource The device and channel(s) to add to the list.

dacClockSource the main DAC clock source

offsetDACClockSource the offset DAC clock source

clockSync channel0 clock routing

Returns:

A pointer to the first channel created.

See also:

CreateAOChannel (p. 417)

2.130.2.7 UeiDaqAPI CUeiAOProtectedChannel* UeiDaq::CUeiSession::CreateAOProtectedChannel (std::string *resource*, tUeiAODACMode *mode*, double *measurementRate*, bool *autoRetry*, double *retryRate*)

Create and add one or more channel to the channel list associated with the session. Protected AO channels are available on certain devices such as the DNA-AO-318 and DNA-AO-316. The amount of current flowing through each output is monitored at the given rate and must stay within the specified range, otherwise the device will automatically open the circuit acting as a breaker. You can specify whether the device should attempt to reestablish the circuit and how often it should try to do so.

Parameters:

resource The device and channel(s) to add to the list.

mode Selects which DAC is connected to the output

measurementRate The monitoring rate. Determines how fast the breaker react after an over or under-limit condition.

autoRetry Specifies whether the device will attempt to reestablish the circuit after an over or under-limit condition

retryRate The number of retries per second.

Returns:

A pointer to the first channel created.

See also:

CreateAIChannel (p. 416), **CreateAOChannel** (p. 417), **CreateDIChannel** (p. 420), **CreateDOChannel** (p. 421), **CreateCIChannel** (p. 423), **CreateCOChannel** (p. 425)

2.130.2.8 UeiDaqAPI CUeiAOProtectedCurrentChannel* UeiDaq::CUeiSession::CreateAOProtectedCurrentChannel (std::string *resource*, tUeiAODACMode *mode*, double *measurementRate*, bool *autoRetry*, double *retryRate*)

Create and add one or more channel to the channel list associated with the session. Protected AO current channels are available on certain devices such as the DNA-AO-318-020 The amount of current flowing through each output is monitored at the given rate and must stay within the specified range, otherwise the device will automatically open the circuit acting as a breaker. You can specify whether the device should attempt to reestablish the circuit and how often it should try to do so.

Parameters:

resource The device and channel(s) to add to the list.

mode Selects which DAC is connected to the output

measurementRate The monitoring rate. Determines how fast the breaker react after an over or under-limit condition.

autoRetry Specifies whether the device will attempt to reestablish the circuit after an over or under-limit condition

retryRate The number of retries per second.

Returns:

A pointer to the first channel created.

See also:

CreateAIChannel (p. 416), **CreateAOChannel** (p. 417), **CreateDIChannel** (p. 420), **CreateDOChannel** (p. 421), **CreateCIChannel** (p. 423), **CreateCOChannel** (p. 425)

2.130.2.9 UeiDaqAPI CUeiSimulatedTCChannel* UeiDaq::CUeiSession::CreateSimulatedTCChannel (std::string *resource*, tUeiThermocoupleType *tcType*, tUeiTemperatureScale *tempScale*, bool *enableCjc*)

Create and add one or more simulated thermocouple channel to the session.

Parameters:

resource The device and channel(s) to add to the list.

tcType The type of thermocouple to simulate.

tempScale The temperature unit used to convert temperature to volts.

enableCjc The cold junction compensation state.

Returns:

A pointer to the first channel created.

2.130.2.10 UeiDaqAPI CUeiSimulatedRTDChannel* UeiDaq::CUeiSession::CreateSimulatedRTDChannel (std::string *resource*, tUeiRTDType *rtdType*, double *rtdNominalResistance*, tUeiTemperatureScale *tempScale*)

Create and add one or more simulated RTD channel to the channel list associated with the session.

Parameters:

resource The device and channel(s) to add to the list.

rtdType The type of RTD connected on the Analog input(s).

rtdNominalResistance The nominal resistance of the RTD at ice point (0 deg Celsius).

tempScale The temperature unit used to convert resistance to temperature.

Returns:

A pointer to the first channel created.

2.130.2.11 UeiDaqAPI CUeiDIChannel* UeiDaq::CUeiSession::CreateDIChannel (std::string *resource*)

Create and add one or more channel to the DI channel list associated with the session. Each channel is associated with a digital input port that groups a predefined number of input lines.

Parameters:

resource The device and channel(s) to add to the list.

Returns:

A pointer to the first channel created.

See also:

CreateAIChannel (p. 416), **CreateAOChannel** (p. 417), **CreateDOChannel** (p. 421), **CreateCIChannel** (p. 423), **CreateCOChannel** (p. 425)

Examples:

DigitalInEvent.cpp.

2.130.2.12 UeiDaqAPI CUeiDIIndustrialChannel* UeiDaq::CUeiSession::CreateDIIndustrialChannel (std::string *resource*, double *lowThreshold*, double *highThreshold*, double *minPulseWidth*)

Create and add one or more channel to the channel list associated with the session. Each channel is associated with a digital input port that groups a predefined number of input lines. Industrial channels are only available on certain DIO devices such as the DNA-DIO-448. You can program the levels at which the input line state change as well as configure a digital filter to eliminate glitches and spikes

Parameters:

resource The device and channel(s) to add to the list.

lowThreshold the low hysteresis threshold

highThreshold the high hysteresis threshold

minPulseWidth the digital filter minimum pulse width in ms. Use 0.0 to disable digital input filter.

Returns:

A pointer to the first channel created.

See also:

CreateAIChannel (p. 416), **CreateAOChannel** (p. 417), **CreateDOChannel** (p. 421), **CreateCIChannel** (p. 423), **CreateCOChannel** (p. 425)

2.130.2.13 UeiDaqAPI CUeiDOChannel* UeiDaq::CUeiSession::CreateDOChannel (std::string *resource*)

Create and add one or more channel to the DO channel list associated with the session. Each channel is associated with a digital output port that groups a predefined number of output lines.

Parameters:

resource The device and channel(s) to add to the list.

Returns:

A pointer to the first channel created.

See also:

CreateAIChannel (p. 416), **CreateAOChannel** (p. 417), **CreateDIChannel** (p. 420), **CreateCIChannel** (p. 423), **CreateCOChannel** (p. 425)

2.130.2.14 UeiDaqAPI CUeiDOIndustrialChannel* UeiDaq::CUeiSession::CreateDOIndustrialChannel (std::string *resource*, tUeiDOPWMMode *pwmMode*, uInt32 *pwmLengthUs*, uInt32 *pwmPeriodUs*, double *pwmDutyCycle*)

Create and add one or more channel to the channel list associated with the session. Each channel is associated with a digital output port that groups a predefined number of output lines. Industrial DO channels are available on certain devices such as the DNx-MF-101 and DNx-DIO-43x. low-to high and high to low transitions can be replaced by pwm providing soft start and soft stop.

Parameters:

resource The device and channel(s) to add to the list.

pwmMode The PWM mode used to soft start and/or stop.

pwmLengthUs The soft start/stop pulse train length in micro-seconds.

pwmPeriodUs The soft start/stop or continuous pulse train period in micro-seconds.

pwmDutyCycle The continuous pulse train duty cycle

Returns:

A pointer to the first channel created.

See also:

CreateAIChannel (p. 416), **CreateAOChannel** (p. 417), **CreateDIChannel** (p. 420), **CreateDOChannel** (p. 421), **CreateCIChannel** (p. 423), **CreateCOChannel** (p. 425)

2.130.2.15 UeiDaqAPI CUeiDOProtectedChannel* UeiDaq::CUeiSession::CreateDOProtectedChannel (std::string *resource*, double *underCurrentLimit*, double *overCurrentLimit*, double *currentSampleRate*, bool *autoRetry*, double *retryRate*)

Create and add one or more channel to the channel list associated with the session. Each channel is associated with a digital output port that groups a predefined number of output lines. Protected channels are available on certain devices such as the DNA-DIO-416 and DNA-DIO-43x. The amount of current flowing through each digital line is monitored at the given rate and must stay within the specified range, otherwise the device will automatically open the circuit acting as a breaker. You can specify whether the device should attempt to reestablish the circuit and how often it should try to do so.

Parameters:

resource The device and channel(s) to add to the list.

underCurrentLimit The minimum amount of current allowed in Amps.

overCurrentLimit The maximum amount of current allowed in Amps.

currentSampleRate The current sampling rate. Determines how fast the breaker react after an over or under-current condition.

autoRetry Specifies whether the device will attempt to reestablish the circuit after an over or under-current condition

retryRate The number of retries per second.

Returns:

A pointer to the first channel created.

See also:

CreateAIChannel (p. 416), **CreateAOChannel** (p. 417), **CreateDIChannel** (p. 420), **CreateDOChannel** (p. 421), **CreateCIChannel** (p. 423), **CreateCOChannel** (p. 425)

2.130.2.16 UeiDaqAPI CUeiCIChannel* UeiDaq::CUeiSession::CreateCIChannel (std::string *resource*, tUeiCounterSource *source*, tUeiCounterMode *mode*, tUeiCounterGate *gate*, Int32 *divider*, Int32 *inverted*)

Create and add one or more channel to the CI channel list associated with the session.

Parameters:

resource The device and channel(s) to add to the list.

source The source of the signal counted or used as a timebase

mode The mode that specify whether the session counts events, measures a pulse width or measures a period on the signal configured as the source

gate The signal that specify whether the counter/timer is on or off

divider The divider used to scale the timebase speed

inverted Specifies whether the signal at the source is inverted before performing the counting operation

Returns:

A pointer to the first channel created.

See also:

CreateAIChannel (p. 416), **CreateAOChannel** (p. 417), **CreateDIChannel** (p. 420), **CreateDOChannel** (p. 421), **CreateCOChannel** (p. 425)

Examples:

BufferedEventCounting.cpp, **EventCount.cpp**, and **MeasureFrequency.cpp**.

2.130.2.17 UeiDaqAPI CUeiQuadratureEncoderChannel* UeiDaq::CUeiSession::CreateQuadratureEncoderChannel (std::string *resource*, uInt32 *initialPosition*, tUeiQuadratureDecodingType *decodingType*, bool *enableZeroIndexing*, tUeiQuadratureZeroIndexPhase *zeroIndexPhase*)

Create and add one or more channel to the CI channel list associated with the session.

Parameters:

resource The device and channel(s) to add to the list.

initialPosition The initial number of pulses when the session starts

decodingType The decoding type 1x, 2x or 4x

enableZeroIndexing Enable or disable resetting the measurement when a zero index event is detected

zeroIndexPhase Specifies the states of A, B and Z inputs that will generate a zero index event

Returns:

A pointer to the first channel created.

See also:

CreateAIChannel (p. 416), **CreateAOChannel** (p. 417), **CreateDIChannel** (p. 420), **CreateDOChannel** (p. 421), **CreateCOChannel** (p. 425)

2.130.2.18 UeiDaqAPI CUeiVRChannel* UeiDaq::CUeiSession::CreateVRChannel (std::string *resource*, tUeiVRMode *mode*)

Create and add one or more channel to the VR channel list associated with the session.

Parameters:

resource The device and channel(s) to add to the list.

mode The VR measurement device can use four different modes to measure velocity, position or direction: Decoder, Timed, NPulses and ZPulse

Returns:

A pointer to the first channel created.

2.130.2.19 UeiDaqAPI CUeiSSIMasterPort* UeiDaq::CUeiSession::CreateSSIMasterPort (std::string *resource*, unsigned int *bps*, unsigned int *wordSize*, bool *enableClock*, bool *enableTermination*, double *pauseTime*, double *transferTimeout*, double *bitUpdateTime*)

Create and add one or more port to the SSI channel list associated with the session.

Parameters:

resource The device and channel(s) to add to the list.

bps The speed of the clock

wordSize The number of bits per word (3 to 32)

enableClock Enable or disable clock output

enableTermination Enable or disable termination resistor

pauseTime Specifies the time delay between two consecutive clock sequences from the master

transferTimeout Specifies the minimum time required by the slave to realise that the data transmission is complete

bitUpdateTime Specifies the minimum time elapsed between retransmissions of the same data

Returns:

A pointer to the first channel created.

2.130.2.20 UeiDaqAPI CUeiSSISlavePort* UeiDaq::CUeiSession::CreateSSISlavePort (std::string *resource*, unsigned int *bps*, unsigned int *wordSize*, bool *enableTransmit*, bool *enableTermination*, double *pauseTime*, double *transferTimeout*, double *bitUpdateTime*)

Create and add one or more port to the SSI channel list associated with the session.

Parameters:

resource The device and channel(s) to add to the list.

bps The speed of the clock

wordSize The number of bits per word (3 to 32)

enableTransmit Enable or disable data output

enableTermination Enable or disable termination resistor

pauseTime Specifies the time delay between two consecutive clock sequences from the master

transferTimeout Specifies the minimum time required by the slave to realise that the data transmission is complete

bitUpdateTime Specifies the minimum time elapsed between retransmissions of the same data

Returns:

A pointer to the first channel created.

2.130.2.21 UeiDaqAPI CUeiCOChannel* UeiDaq::CUeiSession::CreateCOChannel (std::string *resource*, tUeiCounterSource *source*, tUeiCounterMode *mode*, tUeiCounterGate *gate*, uInt32 *tick1*, uInt32 *tick2*, Int32 *divider*, Int32 *inverted*)

Create and add one or more channel to the CO channel list associated with the session.

Parameters:

resource The device and channel(s) to add to the list.

source The timebase used to determine the shape of the signal generated on the counter output

mode The mode that specify whether the session generate a pulse or a pulse train on the counter output

gate The signal that specify whether the counter/timer is on or off

tick1 Specifies how long the output stays low, number of ticks of the signal connected at the source.

tick2 Specifies how long the output stays high, number of ticks of the signal connected at the source.

divider The divider used to scale the timebase speed

inverted Specifies whether the signal at the source is inverted before performing the operation

Returns:

A pointer to the first channel created.

See also:

CreateAIChannel (p. 416), **CreateAOChannel** (p. 417), **CreateDIChannel** (p. 420), **CreateDOChannel** (p. 421), **CreateCIChannel** (p. 423)

Examples:

GeneratePulseTrain.cpp.

2.130.2.22 UeiDaqAPI CUeiTCChannel* UeiDaq::CUeiSession::CreateTCChannel (std::string *resource*, f64 *min*, f64 *max*, tUeiThermocoupleType *tcType*, tUeiTemperatureScale *tempScale*, tUeiColdJunctionCompensationType *cjcType*, f64 *cjcConstant*, std::string *cjcResource*, tUeiAIChannelInputMode *mode*)

Create and add one or more thermocouple channel to the thermocouple channel list associated with the session.

Example:

```
CUeiSession mySession;

mySession.CreateTCChannel("pwrdaq://Dev1/Ai0", 0.0, 1000.0, UeiThermocoupleTypeJ,
                          UeiTemperatureScaleCelsius, UeiCJCTypeConstant, 25.0, "",
                          UeiAIChannelInputModeDifferential);
```

Parameters:

resource The device and channel(s) to add to the list.

min The minimum value you expect to measure.

max The maximum value you expect to measure.

tcType The type of thermocouple connected on the Analog input(s).

tempScale The temperature unit used to convert volts to temperature.

cjcType The cold junction compensation type.

cjcConstant The cold junction compensation constant, this parameter is only used if cjcType is set to UeiCJCTypeConstant.

cjcResource The resource string of the channel on which the cold junction compensation sensor is connected, this parameter is only used if cjcType is set to UeiCJCTypeResource.

mode The input mode used to measure voltage from the sensor

Returns:

A pointer to the first channel created.

See also:

CreateAIChannel (p. 416)

2.130.2.23 UeiDaqAPI CUeiRTDChannel* UeiDaq::CUeiSession::CreateRTDChannel (std::string *resource*, f64 *min*, f64 *max*, tUeiWiringScheme *wiring*, double *twoWiresLeadResistance*, tUeiRTDType *rtdType*, double *rtdNominalResistance*, tUeiTemperatureScale *tempScale*, tUeiAIChannelInputMode *mode*)

Create and add one or more RTD channel to the channel list associated with the session. In two wires mode, only one analog input channel per RTD is required. In four wires mode, two analog

input channels per RTD are required effectively dividing the number of available channels on your device by two. Channels are paired consecutively: (0,1), (2,3), (3,4).... You only need to specify the first channel in the channel list. Read the DNA-STP-AIU manual to learn how to connect your RTD.

Example:

```
CUeiSession mySession;

mySession.CreateRTDChannel("pdna://192.168.100.2/Dev1/Ai0", -100.0, 100.0,
                           UeiFourWires, 0.0,
                           UeiRTDType3850, 100.0,
                           UeiTemperatureScaleCelsius,
                           UeiAIChannelInputModeDifferential);
```

Parameters:

resource The device and channel(s) to add to the list.

min The minimum value you expect to measure.

max The maximum value you expect to measure.

wiring The wiring scheme used to connect the RTD to the acquisition device.

twoWiresLeadResistance The leads resistance in two wires mode.

rtdType The type of RTD connected on the Analog input(s).

rtdNominalResistance The nominal resistance of the RTD at ice point (0 deg Celsius).

tempScale The temperature unit used to convert resistance to temperature.

mode The input mode used to measure voltage from the sensor

Returns:

A pointer to the first channel created.

See also:

CreateAIChannel (p. 416)

2.130.2.24 UeiDaqAPI CUeiResistanceChannel* UeiDaq::CUeiSession::CreateResistanceChannel (std::string *resource*, f64 *min*, f64 *max*, tUeiWiringScheme *wiring*, double *twoWiresLeadResistance*, tUeiAIChannelInputMode *mode*)

Create and add one or more resistance channel to the channel list associated with the session. In two wires mode, only one analog input channel per resistance is required. In four wires mode, two analog input channels per resistance are required effectively dividing the number of available channels on your device by two. Channels are paired consecutively: (0,1), (2,3), (4,5).... You only need to specify the first channel in the channel list. Read the DNA-STP-AIU manual to learn how to connect your resistance.

Example:

```
CUeiSession mySession;

mySession.CreateResistanceChannel("pdna://192.168.100.2/Dev1/Ai0", 0.0, 1000.0,
                                  UeiFourWires, 0.0,
                                  UeiAIChannelInputModeDifferential);
```

Parameters:

resource The device and channel(s) to add to the list.

min The minimum value you expect to measure.

max The maximum value you expect to measure.

wiring The wiring scheme used to connect the resistive sensor to the acquisition device.

twoWiresLeadResistance The lead resistance in two wires mode.

mode The input mode used to measure voltage from the sensor

Returns:

A pointer to the first channel created.

See also:

CreateAIChannel (p. 416)

2.130.2.25 UeiDaqAPI CUeiAIVExChannel* UeiDaq::CUeiSession::CreateAIVExChannel (std::string *resource*, f64 *min*, f64 *max*, tUeiSensorBridgeType *sensorBridgeType*, double *excitationVoltage*, bool *bUseExcitationForScaling*, tUeiAIChannelInputMode *mode*)

Create and add one or more analog input channel to the channel list associated with the session. This will only work with devices that can provide excitation voltage.

Example:

```
CUeiSession mySession;

mySession.CreateAIVExChannel("pwrdaq://Dev1/Ai0", -100.0, 100.0, UeiSensorFullBridge,
                        10.0, true);
```

Parameters:

resource The device and channel(s) to add to the list.

min The minimum value you expect to measure.

max The maximum value you expect to measure.

sensorBridgeType The type of the wheatstone bridge built-in the sensor

excitationVoltage The excitation voltage to output to the sensor

bUseExcitationForScaling Specifies whether the acquired data is divided by the excitation voltage

mode The input mode used to measure voltage from the sensor

Returns:

A pointer to the first channel created.

See also:

CreateAIChannel (p. 416)

2.130.2.26 UeiDaqAPI CUeiAccelChannel* UeiDaq::CUeiSession::CreateAccelChannel (std::string *resource*, f64 *min*, f64 *max*, double *sensorSensitivity*, double *excitationCurrent*, tUeiCoupling *coupling*, bool *enableLowPassFilter*)

Create and add one or more accelerometer channel to the channel list associated with the session. This will only work with devices that can provide excitation current for ICP and IEPE sensors.

Example:

```
CUeiSession mySession;

// Configure channel 0 on device 1 to acquire acceleration measured
// by a sensor with a sensitivity of 24 mV/g, powered by a current
// of 5mA. the gain of the device is adjusted to measure accelerations
// between -10.0g and + 10.0g
mySession.CreateAccelChannel("pwrdaq://Dev1/Ai0", -10.0, 10.0, 24,
                             5.0, UeiCouplingAC, false);
```

Parameters:

resource The device and channel(s) to add to the list.

min The minimum value you expect to measure. The unit depends on the unit of the accelerometer sensitivity. (mV/g will provide measurements in g).

max The maximum value you expect to measure. The unit depends on the unit of the accelerometer sensitivity. (mV/g will provide measurements in g).

sensorSensitivity The sensitivity of the accelerometer. The unit of this parameter specifies the unit of the measurements.

excitationCurrent The excitation current to output to the sensor

coupling The coupling type. AC coupling turns on a 0.1Hz high pass filter.

enableLowPassFilter Turn on or off the low-pass anti-aliasing filter

Returns:

A pointer to the first channel created.

See also:

CreateAIChannel (p. 416)

2.130.2.27 UeiDaqAPI CUeiDMMChannel* UeiDaq::CUeiSession::CreateDMMChannel (std::string *resource*, f64 *range*, tUeiDMMMeasurementMode *measurementMode*)

Create and add one or more analog input channel to the channel list associated with the session. This will only work with DMM devices.

Example:

```
CUeiSession mySession;

mySession.CreateDMMChannel("pdna://192.168.100.2/Dev1/Ai0", 10, UeiDMMModeDCVoltage);
```

Parameters:

resource The device and channel(s) to add to the list.

range The input range. Input signals will need to be within -range/+range interval.

measurementMode The measurement mode configuration

Returns:

A pointer to the first channel created.

See also:

CreateAIChannel (p. 416)

2.130.2.28 UeiDaqAPI CUeiLVDTChannel* UeiDaq::CUeiSession::CreateLVDTChannel (std::string *resource*, f64 *min*, f64 *max*, double *sensorSensitivity*, tUeiLVDTWiringScheme *wiringScheme*, double *excitationVoltage*, double *excitationFrequency*, bool *externalExcitation*)

Create and add one or more LVDT or RVDT channel to the channel list associated with the session. This will only work with devices that can provide excitation waveform to the LVDT or RVDT sensor.

Example:

```
CUeiSession mySession;

// Configure channel 0 on device 1 to acquire position measured
// by a LVDT with a sensitivity of 24 mV/V/mm, powered by a 600Hz
// sine waveform with amplitude of 10.0 VRMS.
// The gain of the device is adjusted to measure positions
// between -10.0mm and + 10.0mm
mySession.CreateLVDTChannel("pdna://192.168.100.2/Dev1/Ai0", -10.0, 10.0, 24,
                            UeiLVDTFiveWires, 10.0, 600.0, false);
```

Parameters:

resource The device and channel(s) to add to the list.

min The minimum value you expect to measure. The unit depends on the unit of the LVDT/RVDT sensitivity. (mV/V/mm will provide measurements in mm).

max The maximum value you expect to measure. The unit depends on the unit of the LVDT/RVDT sensitivity. (mV/V/mm will provide measurements in mm).

sensorSensitivity The sensitivity of the LVDT or RVDT. The unit of this parameter specifies the unit of the measurements.

wiringScheme Specifies whether the LVDT/RVDT is tied to the device using 4 or 5 wires

excitationVoltage The amplitude RMS of the excitation sine waveform to output to the sensor

excitationFrequency The frequency of the excitation sine waveform to output to the sensor

externalExcitation Specifies whether the LVDT is powered by the device or by an external source.

Returns:

A pointer to the first channel created.

See also:

CreateAIChannel (p. 416)

2.130.2.29 UeiDaqAPI CUeiSimulatedLVDTChannel* UeiDaq::CUeiSession::CreateSimulatedLVDTChannel (std::string *resource*, double *sensorSensitivity*, tUeiLVDTWiringScheme *wiringScheme*, double *excitationVoltage*, double *excitationFrequency*)

Create and add one or more simulated LVDT or RVDT channel to the channel list associated with the session.

Example:

```
CUeiSession mySession;

// Configure channel 0 on device 1 to simulate position measurement
// given by a LVDT with a sensitivity of 24 mV/V/mm, powered by a 600Hz
// sine waveform with amplitude of 10.0V RMS.
mySession.CreateSimulatedLVDTChannel("pdna://192.168.100.2/Dev1/Ao0", 24,
                                     UeiLVDTFiveWires, 10.0, 600.0);
```

Parameters:

resource The device and channel(s) to add to the list.

sensorSensitivity The sensitivity of the simulated LVDT or RVDT. The unit of this parameter specifies the unit of the measurements.

wiringScheme Specifies whether the simulated LVDT/RVDT is tied to the device using 4 or 5 wires

excitationVoltage The amplitude RMS of the excitation sine waveform to output to the sensor

excitationFrequency The frequency of the excitation sine waveform to output to the sensor

Returns:

A pointer to the first channel created.

See also:

CreateAOChannel (p. 417)

2.130.2.30 UeiDaqAPI CUeiSynchroResolverChannel* UeiDaq::CUeiSession::CreateSynchroResolverChannel (std::string *resource*, tUeiSynchroResolverMode *mode*, double *excitationVoltage*, double *excitationFrequency*, bool *externalExcitation*)

Create and add one or more Synchro or Resolver channel to the channel list associated with the session. This will only work with devices that can provide excitation waveform to the Synchro or Resolver sensor.

Example:

```
CUeiSession mySession;

// Configure channel 0 on device 1 to acquire position measured
// by a synchro powered by a 600Hz sine waveform with
// amplitude of 10.0 VRMS.
mySession.CreateSynchroResolverChannel("pdna://192.168.100.2/Dev1/Ai0",
                                       UeiSynchroMode, 10.0, 600.0, false);
```

Parameters:

resource The device and channel(s) to add to the list.

mode Specifies whether the input sensor is a synchro or a resolver

excitationVoltage The amplitude RMS of the excitation sine waveform to output to the sensor

excitationFrequency The frequency of the excitation sine waveform to output to the sensor

externalExcitation Specifies whether the sensor is powered by the device or by an external source.

Returns:

A pointer to the first channel created.

See also:

CreateAIChannel (p. 416)

2.130.2.31 UeiDaqAPI CUeiSimulatedSynchroResolverChannel* UeiDaq::CUeiSession::CreateSimulatedSynchroResolverChannel (std::string *resource*, tUeiSynchroResolverMode *mode*, double *excitationVoltage*, double *excitationFrequency*, bool *externalExcitation*)

Create and add one or more simulated synchro or simulated resolver channel to the channel list associated with the session.

Example:

```
CUeiSession mySession;

// Configure channel 0 on device 1 to simulate position measurement
// returned by a synchro powered by a 600Hz sine waveform with
// amplitude of 10.0 VRMS.
mySession.CreateSimulatedSynchroResolverChannel("pdna://192.168.100.2/Dev1/Ao0",
                                                UeiSynchroMode, 10.0, 600.0, false);
```

Parameters:

resource The device and channel(s) to add to the list.

mode Specifies whether the sensor simulated is a synchro or a resolver

excitationVoltage The amplitude RMS of the excitation sine waveform to output to the sensor

excitationFrequency The frequency of the excitation sine waveform to output to the sensor

externalExcitation Specifies whther the sensor is powered by the device or by an external source.

Returns:

A pointer to the first channel created.

See also:

CreateAOChannel (p. 417)

2.130.2.32 UeiDaqAPI CUeiSerialPort* UeiDaq::CUeiSession::CreateSerialPort (std::string *resource*, tUeiSerialPortMode *mode*, tUeiSerialPortSpeed *bitsPerSecond*, tUeiSerialPortDataBits *dataBits*, tUeiSerialPortParity *parity*, tUeiSerialPortStopBits *stopBits*, std::string *termination*)

Create and add one or more serial ports to the session.

Parameters:

resource The device and port(s) to add to the session.

mode The serial port mode: RS-232, RS-485 half and full duplex

bitsPerSecond The number of bits transmitted per second over the serial link

dataBits The number of data bits describing each character

parity The parity scheme used for transmission error detection

stopBits The number of stop bits used to indicate the end of a data message

termination The read operation terminates when the termination string is read from the serial device

Returns:

A pointer to the first port created.

See also:

CreateAIChannel (p. 416), **CreateAOChannel** (p. 417), **CreateDOChannel** (p. 421), **CreateCIChannel** (p. 423), **CreateCOChannel** (p. 425)

2.130.2.33 UeiDaqAPI CUeiHDLCPort* UeiDaq::CUeiSession::CreateHDLCPort (std::string *resource*, tUeiHDLCPortPhysical *physInterface*, uInt32 *bitsPerSecond*, tUeiHDLCPortEncoding *encoding*, tUeiHDLCPortCRCMode *crcMode*, tUeiHDLCPortClockSource *txClockSource*, tUeiHDLCPortClockSource *rxClockSource*)

Create and add one or more HDLC ports to the session.

Parameters:

resource The device and port(s) to add to the session.

physInterface The physical interface: RS-232, RS-485 or RS-422

bitsPerSecond The number of bits transmitted per second

encoding The method used to encode bits on the physical interface

crcMode The method used to calculate each frame CRC

txClockSource The source of the transmitter clock

rxClockSource The source of the receiver clock

Returns:

A pointer to the first port created.

See also:

CreateAIChannel (p. 416), **CreateAOChannel** (p. 417), **CreateDOChannel** (p. 421), **CreateCIChannel** (p. 423), **CreateCOChannel** (p. 425)

2.130.2.34 UeiDaqAPI CUeiCANPort* UeiDaq::CUeiSession::CreateCANPort (std::string *resource*, tUeiCANPortSpeed *bitsPerSecond*, tUeiCANFrameFormat *frameFormat*, tUeiCANPortMode *mode*, uInt32 *acceptanceMask*, uInt32 *acceptanceCode*)

Create and add one or more CAN ports to the session.

Parameters:

resource The device and port(s) to add to the session.

bitsPerSecond The number of bits transmitted per second over the CAN port

frameFormat Specifies the format of frames sent, basic (11 bits ID) or extended (29 bits ID)

mode The operation mode, normal or passive to only listen

acceptanceMask The acceptance mask is used to filter incoming frames. The mask selects which bits within arbitration ID will be used for filtering.

acceptanceCode The acceptance code is used to filter incoming frames. The arbitration ID bits selected by the mask are compared to the code and the frame is rejected if there is any difference. If (mask XOR id) AND code == 0 the frame is accepted.

Returns:

A pointer to the first port created.

See also:

CreateAIChannel (p. 416), **CreateAOChannel** (p. 417), **CreateDOChannel** (p. 421), **CreateCIChannel** (p. 423), **CreateCOChannel** (p. 425)

2.130.2.35 UeiDaqAPI CUeiARINCInputPort* UeiDaq::CUeiSession::CreateARINCInputPort (std::string *resource*, tUeiARINCPortSpeed *bitsPerSecond*, tUeiARINCPortParity *parity*, bool *enableSDIFilter*, uInt32 *SDIMask*)

Create and add one or more ARINC input ports to the session.

Parameters:

resource The device and port(s) to add to the session.

bitsPerSecond The number of bits transmitted per second over the ARINC port

parity The parity used to detect transmission errors.

enableSDIFilter Set to true to enable frame filtering based on their SDI bits.

SDIMask The mask used to match incoming frame SDI bits (only bits 0 and 1 are significant).

Returns:

A pointer to the first port created.

See also:

CreateARINCOutputPort (p. 435)

2.130.2.36 UeiDaqAPI CUeiARINCOutputPort* UeiDaq::CUeiSession::CreateARINCOutputPort (std::string *resource*, tUeiARINCPortSpeed *bitsPerSecond*, tUeiARINCPortParity *parity*)

Create and add one or more ARINC output ports to the session.

Parameters:

resource The device and port(s) to add to the session.

bitsPerSecond The number of bits transmitted per second over the ARINC port

parity The parity used to detect transmission errors.

Returns:

A pointer to the first port created.

See also:

CreateARINCInputPort (p. 434)

2.130.2.37 UeiDaqAPI CUeiMIL1553Port* UeiDaq::CUeiSession::CreateMIL1553Port (std::string *resource*, tUeiMIL1553PortCoupling *coupling*, tUeiMIL1553PortOpMode *portMode*)

Create and add one or more MIL-1553 ports to the session.

Parameters:

resource The device and port(s) to add to the session.

coupling The way how the port is connected to MIL-1553 bus (tranformer, direct, etc.)

portMode The mode of operation (bus monitor, remote terminal, bus controller)

Returns:

A pointer to the first port created.

2.130.2.38 UeiDaqAPI CUeiIRIGTimeKeeperChannel* UeiDaq::CUeiSession::CreateIRIGTimeKeeperChannel (std::string *resource*, tUeiIRIGTimeKeeper1PPSSource *source*, bool *autoFollow*)

Create and add one or more IRIG time keeper channels to the session.

Parameters:

resource The device and port(s) to add to the session.

source The 1 PPS signal source

autoFollow Enable or disable auto follow

Returns:

A pointer to the first channel created.

2.130.2.39 UeiDaqAPI CUeiIRIGInputChannel* UeiDaq::CUeiSession::CreateIRIGInputChannel (std::string *resource*, tUeiIRIGDecoderInputType *inputType*, tUeiIRIGTimeCodeFormat *timeCodeFormat*)

Create and add one or more IRIG input channels to the session.

Parameters:

resource The device and port(s) to add to the session.

inputType The input where the timecode signal is connected

timeCodeFormat The time code format used by the input signal

Returns:

A pointer to the first channel created.

2.130.2.40 UeiDaqAPI CUeiIRIGOutputChannel* UeiDaq::CUeiSession::CreateIRIGOutputChannel (std::string *resource*, tUeiIRIGTimeCodeFormat *timeCodeFormat*)

Create and add one or more IRIG output channels to the session.

Parameters:

resource The device and port(s) to add to the session.

timeCodeFormat The time code format to use for the generated signal

Returns:

A pointer to the first channel created.

2.130.2.41 UeiDaqAPI CUeiIRIGDOTTLChannel* UeiDaq::CUeiSession::CreateIRIGDOTTLChannel (std::string *resource*, tUeiIRIGDOTTLSource *source0*, tUeiIRIGDOTTLSource *source1*, tUeiIRIGDOTTLSource *source2*, tUeiIRIGDOTTLSource *source3*)

Create and add one or more IRIG DO TTL channels to the session.

Parameters:

resource The device and port(s) to add to the session.

source0 The source used to generate TTL pulses on output 0

source1 The source used to generate TTL pulses on output 1

source2 The source used to generate TTL pulses on output 2

source3 The source used to generate TTL pulses on output 3

Returns:

A pointer to the first channel created.

2.130.2.42 UeiDaqAPI CUeiSync1PPSPort* UeiDaq::CUeiSession::CreateSync1PPSPort (std::string *resource*, tUeiSync1PPSMode *mode*, tUeiSync1PPSSource *source*, double *EMOutputRate*)

The 1PPS synchronization port uses an ADPLL and an event module to produce a clock at an arbitrary rate Multiple racks can easily be synchronized when they use the same 1PPS synchronization clock The ADPLL+event module on each rack will produce synchronized scan clocks

The source of the 1PPS sync clock can be internal, external or generated from NTP The 1PPS clock is routed via one of the four internal sync lines to the ADPLL (adaptive digital PLL) The ADPLL locks on the 1PPS and outputs its own 1PPS that is an average of the original 1PPS clock The ADPLL can maintain its 1PPS output even if the original 1PPS clock gets disconnected The ADPLL 1PPS output is routed via one of the four sync lines to the event module The ADPLL 1PPS output can also be routed to the sync connector for sharing with other racks/cubes The event module produces a user selectable number of pulses upon every 1PPS pulse coming from the ADPLL. The event module clock output is routed to the Input layers via one of the remaining sync lines

Parameters:

resource The IO module IP address.

mode The mode used to obtain the reference 1PPS (existing 1PPS clock, NTP or 1588)

source The source of the 1PPS refenrece clock

EMOutputRate The rate of the resulting clock (synchronized with the 1PPS reference)

Returns:

A pointer to the synchronization port.

2.130.2.43 UeiDaqAPI CUeiCSDBPort* UeiDaq::CUeiSession::CreateCSDBPort (std::string *resource*, int *bps*, int *parity*, int *blockSize*, int *numberOfMessages*, int *interByteDelayUs*, int *interBlockDelayUs*, int *framePeriodUs*)

Create and add one or more CSDB ports to the session.

Parameters:

resource The device and port(s) to add to the session.

bps The CSDB port speed in bits per second (12500 or 50000)

parity The parity (0 for even >0 for odd)

blockSize The number of bytes per block (including address and status bytes)

numberOfMessages The number of message blocks per frame

interByteDelayUs The delay between bytes within a block

interBlockDelayUs The delay between blocks within a frame

framePeriodUs The period at which cyclic frames are emitted

Returns:

A pointer to the first port created.

2.130.2.44 UeiDaqAPI CUeiMuxPort* UeiDaq::CUeiSession::CreateMuxPort (std::string *resource*, bool *breakBeforeMake*)

Create and add a MUX port to the session.

Parameters:

resource The device and port to add to the session.

breakBeforeMake True to enable break before make, false otherwise

Returns:

A pointer to the first port created.

2.130.2.45 UeiDaqAPI CUeiI2CMasterPort* UeiDaq::CUeiSession::CreateI2CMasterPort (std::string *resource*, tUeiI2CPortSpeed *bitRate*, tUeiI2CTTLLevel *ttlLevel*, bool *enableSecureShell*)

Create and add an I2C master port to the session.

Parameters:

resource The device and port to add to the session.

bitRate the clock speed (bits per second)

ttlLevel the TTL level used by clock and data signals

enableSecureShell True to enable secure shell feature, false otherwise

Returns:

A pointer to the first port created.

2.130.2.46 UeiDaqAPI CUeiI2CSlavePort* UeiDaq::CUeiSession::CreateI2CSlavePort (std::string *resource*, tUeiI2CTTLLevel *ttlLevel*, tUeiI2CSlaveAddressWidth *addressWidth*, int *address*)

Create and add an I2C slave port to the session.

Parameters:

resource The device and port to add to the session.

ttlLevel the TTL level used by clock and data signals

addressWidth specify the number of bits used to encode this slave address

address the address of this slave port

Returns:

A pointer to the first port created.

2.130.2.47 UeiDaqAPI void UeiDaq::CUeiSession::CreateInfoSession (std::string *resource*)

This session is useful to retrieve information about a device or directly access its registers.

Parameters:

resource The device.

2.130.2.48 UeiDaqAPI void UeiDaq::CUeiSession::ConfigureTimingForSimpleIO ()

Configure the timing object for simple IO. In this mode scans are acquired/generated one at a time. The pace of the acquisition/generation is entirely determined by the speed at which your software is calling the read/write functions.

See also:

ConfigureTimingForBufferedIO (p. 439), **ConfigureTimingForEdgeDetection** (p. 440), **ConfigureTimingForTimeSequencing** (p. 441), **ConfigureTimingForDataMappingIO** (p. 440)

Examples:

AnalogInSingle.cpp, **AnalogOutSingle.cpp**, **EventCount.cpp**, **GeneratePulseTrain.cpp**, and **MeasureFrequency.cpp**.

2.130.2.49 UeiDaqAPI void UeiDaq::CUeiSession::ConfigureTimingForBufferedIO (int *samplePerChannel*, tUeiTimingClockSource *clkSource*, double *rate*, tUeiDigitalEdge *edge*, tUeiTimingDuration *duration*)

Configure the timing object for buffered mode using hardware timing

Parameters:

samplePerChannel The number of samples to acquire if duration is one-shot.

clkSource The source of the clock.

rate The frequency of the scan clock in Hz.

edge The edge of the clock that triggers the scan acquisition: rising or falling.

duration The duration of the acquisition/generation, continuous or single-shot.

See also:

ConfigureTimingForSimpleIO (p. 439), **ConfigureTimingForEdgeDetection** (p. 440), **ConfigureTimingForTimeSequencing** (p. 441)

Examples:

AnalogInBuffered.cpp, **AnalogInBufferedAsync.cpp**, **AnalogInOneShot_Trig_Ex.cpp**, **AnalogOutBuffered.cpp**, **AnalogOutBufferedAsync.cpp**, and **BufferedEventCounting.cpp**.

2.130.2.50 UeiDaqAPI void UeiDaq::CUeiSession::ConfigureTimingForDataMappingIO (tUeiTimingClockSource *clkSource*, double *rate*)

Configure the timing object for direct data mapping mode using hardware timing

Parameters:

clkSource The source of the clock.

rate The refresh rate of the data map.

See also:

ConfigureTimingForSimpleIO (p. 439), **ConfigureTimingForEdgeDetection** (p. 440), **ConfigureTimingForTimeSequencing** (p. 441)

2.130.2.51 UeiDaqAPI void UeiDaq::CUeiSession::ConfigureTimingForEdgeDetection (tUeiDigitalEdge *edge*)

Configure the timing object for digital line state change detection The line on which the state change is sensed is specified by calling the SetEdgeMask() method on the DIChannel object.

Parameters:

edge The edge of the signal that triggers the state change detection event.

See also:

SetEdgeMask, **ConfigureTimingForSimpleIO** (p. 439), **ConfigureTimingForBufferedIO** (p. 439), **ConfigureTimingForTimeSequencing** (p. 441)

Examples:

DigitalInEvent.cpp.

2.130.2.52 UeiDaqAPI void UeiDaq::CUeiSession::ConfigureTimingForAsynchronousIO (int *watermark*, int *periodUs*, int *noActivityTimeoutUs*, int *maxDataSize*)

Configure the timing object for asynchronous FIFO based I/O Data is read/written when the input/output FIFO reaches the specified watermark level, when the periodic timer expires or when the no activity timeout expires (whichever happens first) Set watermark to -1 to disable watermark events Set period to -1 to disable periodic timer events Set no activity timeout to -1 to disable no activity events

Parameters:

watermark The watermark level used to configure FIFO asynchronous event. (-1) to disable

periodUs The period at which an event is fired when the FIFO level stays below watermark for too long (-1) to disable)

noActivityTimeoutUs The maximum time to wait for any activity (-1) disable

maxDataSize The maximum number of values returned along with an event

2.130.2.53 UeiDaqAPI void UeiDaq::CUeiSession::ConfigureTimingForTimeSequencing (int *samplePerChannel*, tUeiTimingDuration *duration*)

In Time sequencing mode, the timing information is built-in the data buffer written to the board

Parameters:

samplePerChannel The number of samples to acquire if duration is one-shot.

duration The duration of the acquisition/generation, continuous or single-shot.

See also:

ConfigureTimingForSimpleIO (p. 439), **ConfigureTimingForEdgeDetection** (p. 440), **ConfigureTimingForBufferedIO** (p. 439)

2.130.2.54 UeiDaqAPI void UeiDaq::CUeiSession::ConfigureTimingForMessagingIO (int *bufferSize*, double *refreshRate*)

Configure the session to do messaging IOs. This timing mode is only available for Serial, CAN, ARINC-429 and MIL-1553 bus sessions. The Serial, CAN, ARINC-429 and MIL-1553 bus devices can be programmed to wait for a certain number of messages to be received before notifying the session. It is also possible to program the maximum amount of time to wait for the specified number of messages before notifying the session.

For serial sessions a message is simply a byte, for CAN sessions a message is a CAN frame represented with the data structure tUeiCANFrame.

Parameters:

bufferSize The miminum number of messages to buffer before notifying the session

refreshRate The rate at which the device notifies the session that messages have been received. Set the rate to 0 to be notified immediately when a message is received.

See also:

ConfigureTimingForBufferedIO (p. 439), **ConfigureTimingForEdgeDetection** (p. 440), **ConfigureTimingForTimeSequencing** (p. 441)

2.130.2.55 UeiDaqAPI void UeiDaq::CUeiSession::ConfigureTimingForVMapIO (double *rate*)

Configure the timing object forVMAP mode VMap mode provides access to a device's FIFO without any buffering

Parameters:

rate The rate at which data is clocked in or out of the FIFO (relevant only for clocked I/Os such as AI, AO, DI and DO).

See also:

ConfigureTimingForSimpleIO (p. 439), **ConfigureTimingForEdgeDetection** (p. 440), **ConfigureTimingForTimeSequencing** (p. 441)

2.130.2.56 UeiDaqAPI void UeiDaq::CUeiSession::ConfigureStartDigitalTrigger (tUeiTriggerSource *source*, tUeiDigitalEdge *edge*)

Configure the digital trigger condition that will start the session

Parameters:

source The trigger source descriptor

edge The edge of the trigger signal that starts the session.

2.130.2.57 UeiDaqAPI void UeiDaq::CUeiSession::ConfigureStopDigitalTrigger (tUeiTriggerSource *source*, tUeiDigitalEdge *edge*)

Configure the digital trigger condition that will stop the session

Parameters:

source The trigger source descriptor

edge The edge of the trigger signal that stops the session.

2.130.2.58 UeiDaqAPI void UeiDaq::CUeiSession::ConfigureAnalogSoftwareTrigger (tUeiTriggerAction *action*, tUeiTriggerCondition *condition*, Int32 *triggerChannel*, f64 *level*, f64 *hysteresis*, Int32 *preTriggerScans*)

The analog software trigger looks at every sample acquired on the specified channel until the trigger condition is met. Once the trigger condition is met the application can read the acquired scans in a buffer. The trigger level and hysteresis are specified in the same unit as the measurements.

Parameters:

action The action to execute when the trigger condition is met.

condition The condition for the trigger to occur.

triggerChannel The channel to monitor for the trigger condition.

level The scaled value that defines the trigger threshold.

hysteresis The scaled size of the hysteresis window.

preTriggerScans The number of scans to save before the trigger occurs.

2.130.2.59 UeiDaqAPI void UeiDaq::CUeiSession::ConfigureSignalTrigger (tUeiTriggerAction *action*, std::string *signal*)

Configure the trigger signal that will start or stop the session The trigger signal is device dependent, it can be a physical line on a backplane (such as a PXI trigger) or a software signal used to synchronize multiple devices (for example starting all layers in a PowerDNA cube simultaneously).

Parameters:

action The action to execute when the signal occurs.

signal The signal that will transmit the trigger

2.130.2.60 UeiDaqAPI void UeiDaq::CUeiSession::AddEventListener (IUeiEventListener * *pListener*)

Add an event listener that will be called each time an event is fired.

Parameters:

pListener User implementation of **IUeiEventListener** (p. 549) interface.

2.130.2.61 UeiDaqAPI void UeiDaq::CUeiSession::Start ()

Start the session immediately of after the trigger condition occurred.

See also:

Stop (p. 444) **CleanUp** (p. 444) **IsRunning** (p. 443) **WaitUntilDone** (p. 443)

Examples:

AnalogInBuffered.cpp, **AnalogInBufferedAsync.cpp**, **AnalogInOneShot_Trig_Ex.cpp**, **AnalogInSingle.cpp**, **AnalogOutBuffered.cpp**, **AnalogOutBufferedAsync.cpp**, **BufferedEventCounting.cpp**, **DigitalInEvent.cpp**, **EventCount.cpp**, **GeneratePulseTrain.cpp**, and **MeasureFrequency.cpp**.

2.130.2.62 UeiDaqAPI bool UeiDaq::CUeiSession::IsRunning ()

Check whether the session is in the running state.

Returns:

true if the session is still running and false otherwise

See also:

Start (p. 443) **Stop** (p. 444) **CleanUp** (p. 444) **WaitUntilDone** (p. 443)

Examples:

AnalogOutBuffered.cpp.

2.130.2.63 UeiDaqAPI bool UeiDaq::CUeiSession::WaitUntilDone (int *timeout*)

Wait for the specified time until the session is stopped.

Parameters:

timeout Maximum amount of ms this call will wait.

Returns:

true if the session is finished, false otherwise

See also:

Start (p. 443) **Stop** (p. 444) **CleanUp** (p. 444) **IsRunning** (p. 443)

2.130.2.64 UeiDaqAPI void UeiDaq::CUeiSession::Stop ()

Stop the session immediately and wait for the session to be in the stopped state before returning, once it returned you can restart the session immediately with **Start()** (p. 443) without having to reconfigure channels. timing or triggers.

See also:

Start (p. 443) **CleanUp** (p. 444) **IsRunning** (p. 443) **WaitUntilDone** (p. 443)

Examples:

AnalogInBuffered.cpp, **AnalogInBufferedAsync.cpp**, **AnalogInOneShot_Trig_Ex.cpp**, **AnalogInSingle.cpp**, **AnalogOutBuffered.cpp**, **AnalogOutBufferedAsync.cpp**, **BufferedEventCounting.cpp**, **DigitalInEvent.cpp**, **EventCount.cpp**, **GeneratePulseTrain.cpp**, and **MeasureFrequency.cpp**.

2.130.2.65 UeiDaqAPI int UeiDaq::CUeiSession::Pause (int *port*)

Pause the specified messaging port.

Parameters:

port The port to pause

Returns:

the number of elements that are pending and might need to be resent after the pause

2.130.2.66 UeiDaqAPI void UeiDaq::CUeiSession::Resume (int *port*)

Pause the specified messaging port.

Parameters:

port The port to resume

2.130.2.67 UeiDaqAPI void UeiDaq::CUeiSession::CleanUp ()

Stop and Clean-up the session. All session parameters are lost and you need to reconfigure channels, timing or triggers if you want to reuse the same session object.

See also:

Stop (p. 444) **Start** (p. 443) **IsRunning** (p. 443) **WaitUntilDone** (p. 443)

2.130.2.68 UeiDaqAPI void UeiDaq::CUeiSession::EnableAutoReStart (bool *enable*)

Enabling auto-restart will automatically stop/start a running session once it is detected that the device was reset (uually because of a power-cycle).

Parameters:

enable true to auto-restart, false otherwise

2.130.2.69 UeiDaqAPI CUeiDevice* UeiDaq::CUeiSession::GetDevice ()

Get a pointer to the device object associated with the session.

Returns:

A pointer to a **CUeiDevice** (p. 188) object that can be later used to retrieve information about the device associated to the session.

Examples:

MeasureFrequency.cpp.

2.130.2.70 UeiDaqAPI CUeiTiming* UeiDaq::CUeiSession::GetTiming ()

Get a pointer to the timing object associated with the session.

Returns:

A pointer to the **CUeiTiming** (p. 517) object that is used to configure timing informations.

Examples:

AnalogInOneShot_Trig_Ex.cpp, and **DigitalInEvent.cpp**.

2.130.2.71 UeiDaqAPI CUeiDataStream* UeiDaq::CUeiSession::GetDataStream ()

Get a pointer to the stream object associated with the session.

Returns:

A pointer to the **CUeiDataStream** (p. 176) object that is used to access data.

Examples:

AnalogInBuffered.cpp, **AnalogInBufferedAsync.cpp**, **AnalogInOneShot_Trig_Ex.cpp**, **AnalogInSingle.cpp**, **AnalogOutBuffered.cpp**, **AnalogOutBufferedAsync.cpp**, **AnalogOutSingle.cpp**, **BufferedEventCounting.cpp**, **DigitalInEvent.cpp**, **EventCount.cpp**, **GeneratePulseTrain.cpp**, and **MeasureFrequency.cpp**.

2.130.2.72 UeiDaqAPI CUeiTrigger* UeiDaq::CUeiSession::GetStartTrigger ()

Get a pointer to the start trigger object associated with the session.

Returns:

A pointer to the **CUeiTrigger** (p. 533) object that is used to access triggers data.

2.130.2.73 UeiDaqAPI CUeiTrigger* UeiDaq::CUeiSession::GetStopTrigger ()

Get a pointer to the stop trigger object associated with the session.

Returns:

A pointer to the **CUeiTrigger** (p. 533) object that is used to access triggers data.

2.130.2.74 UeiDaqAPI int UeiDaq::CUeiSession::GetNumberOfChannels ()

Get the number of channels in the channel list.

Returns:

number of channels.

Examples:

AnalogInBuffered.cpp, **AnalogInBufferedAsync.cpp**, **AnalogInSingle.cpp**, **AnalogOutBuffered.cpp**, **AnalogOutBufferedAsync.cpp**, **AnalogOutSingle.cpp**, and **MeasureFrequency.cpp**.

2.130.2.75 UeiDaqAPI CUeiChannel* UeiDaq::CUeiSession::GetChannel (int *index*)

Get the channel object specified by its index in the channel list.

Parameters:

index index of the channel object in the channel list

Returns:

A pointer to the channel object.

Examples:

MeasureFrequency.cpp.

2.130.2.76 UeiDaqAPI CUeiChannel* UeiDaq::CUeiSession::GetChannelById (int *id*)

Get the channel object specified by its physical channel id.

Parameters:

id physical id of the channel object

Returns:

A pointer to the channel object.

2.130.2.77 UeiDaqAPI void UeiDaq::CUeiSession::SetCustomProperty (std::string *property*, int *valueSize*, void * *value*)

Set a custom property on the device or subsystem associated with this session Custom properties are device dependent, refer to the device user manual to learn about the properties supported by your device

Parameters:

property string containing the name of the property to set

valueSize size in bytes of the memory block containing the property value

value pointer to the memory block containing the property value

2.130.2.78 UeiDaqAPI void UeiDaq::CUeiSession::SetCustomProperty (std::string *property*, int *value*)

Set a custom property on the device or subsystem associated with this session Custom properties are device dependent, refer to the device user manual to learn about the properties supported by your device

Parameters:

property string containing the name of the property to set

value contain the new property value

2.130.2.79 UeiDaqAPI void UeiDaq::CUeiSession::SetCustomProperty (std::string *property*, double *value*)

Set a custom property on the device or subsystem associated with this session Custom properties are device dependent, refer to the device user manual to learn about the properties supported by your device

Parameters:

property string containing the name of the property to set

value contain the new property value

2.130.2.80 UeiDaqAPI void UeiDaq::CUeiSession::SetCustomProperty (std::string *property*, int *arraySize*, int * *valueArray*)

Set a custom property on the device or subsystem associated with this session Custom properties are device dependent, refer to the device user manual to learn about the properties supported by your device

Parameters:

property string containing the name of the property to set

arraySize number of integers in the array

valueArray contain the new property value

2.130.2.81 UeiDaqAPI void UeiDaq::CUeiSession::SetCustomProperty (std::string *property*, int *arraySize*, double * *valueArray*)

Set a custom property on the device or subsystem associated with this session Custom properties are device dependent, refer to the device user manual to learn about the properties supported by your device

Parameters:

property string containing the name of the property to set

arraySize number of doubles in the array

valueArray contain the new property value

2.130.2.82 UeiDaqAPI void UeiDaq::CUeiSession::GetCustomProperty (std::string *property*, int *valueSize*, void * *value*)

Get a custom property on the device or subsystem associated with this session Custom properties are device dependent, refer to the device user manual to learn about the properties supported by your device

Parameters:

property string containing the name of the property to get

valueSize size in bytes of the memory block receiving the property value

value pointer to the memory block receiving the property value

2.130.2.83 UeiDaqAPI void UeiDaq::CUeiSession::GetCustomProperty (std::string *property*, int * *value*)

Get a custom property on the device or subsystem associated with this session Custom properties are device dependent, refer to the device user manual to learn about the properties supported by your device

Parameters:

property string containing the name of the property to get

value pointer to the current property value

2.130.2.84 UeiDaqAPI void UeiDaq::CUeiSession::GetCustomProperty (std::string *property*, double * *value*)

Get a custom property on the device or subsystem associated with this session Custom properties are device dependent, refer to the device user manual to learn about the properties supported by your device

Parameters:

property string containing the name of the property to get

value pointer to the current property value

2.130.2.85 UeiDaqAPI void UeiDaq::CUeiSession::GetCustomProperty (std::string *property*, int *arraySize*, int * *valueArray*)

Get a custom property on the device or subsystem associated with this session Custom properties are device dependent, refer to the device user manual to learn about the properties supported by your device

Parameters:

property string containing the name of the property to get

arraySize maximum number of elements that can be stored in value array

valueArray pointer to the current property value

2.130.2.86 UeiDaqAPI void UeiDaq::CUeiSession::GetCustomProperty (std::string *property*, int *arraySize*, double * *valueArray*)

Get a custom property on the device or subsystem associated with this session Custom properties are device dependent, refer to the device user manual to learn about the properties supported by your device

Parameters:

property string containing the name of the property to get

arraySize maximum number of elements that can be stored in value array

valueArray pointer to the current property value

2.130.2.87 UeiDaqAPI tUeiSessionType UeiDaq::CUeiSession::GetType ()

Get the session type of this session. The session type is set by the first channel added to the session.

Returns:

the session type

2.130.2.88 UeiDaqAPI int UeiDaq::CUeiSession::GetSubsystem ()

Get the subsystem index of this session.

Returns:

the subsystem index

2.130.2.89 UeiDaqAPI void UeiDaq::CUeiSession::LoadFromXml (std::string *xmlSettings*)

Load Session settings from an XML stream, session is ready to be started

Parameters:

xmlSettings String containing the session settings in XML format

2.130.2.90 UeiDaqAPI void UeiDaq::CUeiSession::LoadFromFile (std::string *sessionFile*)

Load Session settings from a file, session is ready to be started

Parameters:

sessionFile String containing the session settings file path

The documentation for this class was generated from the following file:

- UeiSession.h

2.131 UeiDaq::CUeiSimulatedLVDTChannel Class Reference

Manages settings for each LVDT/RVDT output channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiAOChannel**.

Public Member Functions

- UeiDaqAPI **CUeiSimulatedLVDTChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiSimulatedLVDTChannel** ()
  Destructor.
- UeiDaqAPI **tUeiLVDTWiringScheme GetWiringScheme** ()
  Get the wiring scheme.
- UeiDaqAPI void **SetWiringScheme** (**tUeiLVDTWiringScheme** wiring)
  Set the wiring scheme.
- UeiDaqAPI double **GetExcitationVoltage** ()
  Get the excitation RMS voltage.
- UeiDaqAPI void **SetExcitationVoltage** (double vex)
  Set the excitation RMS voltage.
- UeiDaqAPI double **GetExcitationFrequency** ()
  Get the excitation frequency.
- UeiDaqAPI void **SetExcitationFrequency** (double fex)
  Set the excitation frequency.
- UeiDaqAPI double **GetSensorSensitivity** ()
  Get the sensor sensitivity.
- UeiDaqAPI void **SetSensorSensitivity** (double sensitivity)
  Set the sensor sensitivity.
- UeiDaqAPI double **GetOffset** ()
  Get the fine-tuning offset.
- UeiDaqAPI void **SetOffset** (double offset)
  Set the fine-tuning offset.
- UeiDaqAPI double **GetGain** ()
  Get the fine-tuning gain.
- UeiDaqAPI void **SetGain** (double gain)

Set the fine-tuning gain.

- UeiDaqAPI bool **IsExternalAmplitudeAutoFollowEnabled** ()
  Get external amplitude auto follow.
- UeiDaqAPI void **EnableExternalAmplitudeAutoFollow** (bool enable)
  Set external amplitude auto follow.

2.131.1 Detailed Description

Manages settings for each LVDT/RVDT output channel

2.131.2 Member Function Documentation

2.131.2.1 UeiDaqAPI tUeiLVDTWiringScheme UeiDaq::CUeiSimulatedLVDTChannel::GetWiringScheme ()

Get the current wiring scheme used for this channel.

Returns:

The current wiring scheme.

See also:

SetWiringScheme (p. 452)

2.131.2.2 UeiDaqAPI void UeiDaq::CUeiSimulatedLVDTChannel::SetWiringScheme (tUeiLVDTWiringScheme *wiring*)

Specifies the wiring scheme used to connect.

Parameters:

wiring The new wiring scheme.

See also:

GetWiringScheme (p. 452)

2.131.2.3 UeiDaqAPI double UeiDaq::CUeiSimulatedLVDTChannel::GetExcitationVoltage ()

Get the excitation RMS voltage configured for the channel.

Returns:

The excitation voltage.

See also:

SetExcitationVoltage (p. 453)

2.131.2.4 UeiDaqAPI void UeiDaq::CUeiSimulatedLVDTChannel::SetExcitationVoltage (double *vex*)

Set the excitation RMS for this channel.

Parameters:

vex The excitation voltage

See also:

GetExcitationVoltage (p. 452)

2.131.2.5 UeiDaqAPI double UeiDaq::CUeiSimulatedLVDTChannel::GetExcitation-Frequency ()

Get the excitation frequency configured for the channel.

Returns:

The excitation frequency.

See also:

SetExcitationFrequency (p. 453)

2.131.2.6 UeiDaqAPI void UeiDaq::CUeiSimulatedLVDTChannel::SetExcitationFrequency (double *fex*)

Set the excitation frequency for this channel.

Parameters:

fex The excitation frequency

See also:

GetExcitationFrequency (p. 453)

2.131.2.7 UeiDaqAPI double UeiDaq::CUeiSimulatedLVDTChannel::GetSensorSensitivity ()

Get the simulated sensor sensitivity, it usually is in mVolts/V/[EU]

Returns:

the current sensitivity.

See also:

SetSensorSensitivity (p. 454)

2.131.2.8 UeiDaqAPI void UeiDaq::CUeiSimulatedLVDTChannel::SetSensorSensitivity (double *sensitivity*)

Set the simulated sensor sensitivity, it usually is in mVolts/V/[EU]

Parameters:

sensitivity The new sensitivity

See also:

GetSensorSensitivity (p. 453)

2.131.2.9 UeiDaqAPI double UeiDaq::CUeiSimulatedLVDTChannel::GetOffset ()

Get the fine-tuning offset

Returns:

the current offset.

See also:

SetOffset (p. 454)

2.131.2.10 UeiDaqAPI void UeiDaq::CUeiSimulatedLVDTChannel::SetOffset (double *offset*)

Set the fine-tuning offset

Parameters:

offset The new offset

See also:

GetOffset (p. 454)

2.131.2.11 UeiDaqAPI double UeiDaq::CUeiSimulatedLVDTChannel::GetGain ()

Get the fine-tuning gain

Returns:

the current fain.

See also:

SetGain (p. 455)

2.131.2.12 UeiDaqAPI void UeiDaq::CUeiSimulatedLVDTChannel::SetGain (double *gain*)

Set the fine-tuning gain

Parameters:

gain The new gain

See also:

GetGain (p. 454)

2.131.2.13 UeiDaqAPI bool UeiDaq::CUeiSimulatedLVDTChannel::IsExternalAmplitudeAutoFollowEnabled ()

Simulated LVDT signals amplitude follows external excitation amplitude by default. This setting overrides default amplitude with amplitude specified by excitation voltage setting.

Returns:

curent external amplitude setting

2.131.2.14 UeiDaqAPI void UeiDaq::CUeiSimulatedLVDTChannel::EnableExternalAmplitudeAutoFollow (bool *enable*)

Simulated LVDT signals amplitude follows external excitation amplitude by default. This setting overrides default amplitude with amplitude specified by excitation voltage setting.

Parameters:

enable set new external amplitude setting

The documentation for this class was generated from the following file:

- UeiChannel.h

2.132 UeiDaq::CUeiSimulatedRTDChannel Class Reference

Manages settings for each simulated RTD output channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiAOChannel**.

Public Member Functions

- UeiDaqAPI **CUeiSimulatedRTDChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiSimulatedRTDChannel** ()
  Destructor.
- UeiDaqAPI **tUeiRTDType GetRTDType** ()
  Get the simulated RTD type.
- UeiDaqAPI void **SetRTDType** (**tUeiRTDType** type)
  Set the simulated RTD type.
- UeiDaqAPI double **GetRTDNominalResistance** ()
  Get the simulated RTD nominal resistance.
- UeiDaqAPI void **SetRTDNominalResistance** (double r0)
  Set the simulated RTD nominal resistance.
- UeiDaqAPI double **GetCoefficientA** ()
  Get the Callendar Van-Dusen coefficient A.
- UeiDaqAPI void **SetCoefficientA** (double A)
  Set the Callendar Van-Dusen coefficient A.
- UeiDaqAPI double **GetCoefficientB** ()
  Get the Callendar Van-Dusen coefficient B.
- UeiDaqAPI void **SetCoefficientB** (double B)
  Set the Callendar Van-Dusen coefficient B.
- UeiDaqAPI double **GetCoefficientC** ()
  Get the Callendar Van-Dusen coefficient C.
- UeiDaqAPI void **SetCoefficientC** (double C)
  Set the Callendar Van-Dusen coefficient C.
- UeiDaqAPI **tUeiTemperatureScale GetTemperatureScale** ()
  Get the temperature scale.
- UeiDaqAPI void **SetTemperatureScale** (**tUeiTemperatureScale** tempScale)

Set the temperature scale.

- UeiDaqAPI bool **IsCircuitBreakerEnabled** ()

  Determines whether the CB is currently protecting the channel.

- UeiDaqAPI void **EnableCircuitBreaker** (bool enable)

  Enable or Disable channel protection.

- UeiDaqAPI double **GetCircuitBreakerCurrentLimit** ()

  Get the circuit breaker maximum current limit.

- UeiDaqAPI void **SetCircuitBreakerCurrentLimit** (double currentLimit)

  Set the circuit breaker maximum current limit.

- UeiDaqAPI double **GetCircuitBreakerTemperatureLimit** ()

  Get the maximum circuit breaker temperature limit.

- UeiDaqAPI void **SetCircuitBreakerTemperatureLimit** (double temperatureLimit)

  Set the maximum circuit breaker temperature limit.

- UeiDaqAPI **uInt32 GetOverUnderCount** (void)

  Get the over/under count.

- UeiDaqAPI void **SetOverUnderCount** (**uInt32** overUnderCount)

  Set the over/under count.

2.132.1 Detailed Description

Manages settings for each simulated RTD output channel

2.132.2 Member Function Documentation

2.132.2.1 UeiDaqAPI tUeiRTDType UeiDaq::CUeiSimulatedRTDChannel::GetRTDType ()

RTD sensors are specified using the "alpha" (a) constant It is also known as the temperature coefficient of resistance, and symbolizes the resistance change factor per degree of temperature change. The RTD type is used to select the proper coefficients A, B and C for the Callendar Van-Dusen equation used to convert temperature to resistance.

Returns:

The simulated RTD type.

See also:

SetRTDType (p. 458)

2.132.2.2 UeiDaqAPI void UeiDaq::CUeiSimulatedRTDChannel::SetRTDType (tUeiRTDType *type*)

RTD sensors are specified using the "alpha" (a) constant It is also known as the temperature coefficient of resistance, and symbolizes the resistance change factor per degree of temperature change. The RTD type is used to select the proper coefficients A, B and C for the Callendar Van-Dusen equation used to convert temperature to resistance.

Parameters:

type The simulated RTD type

See also:

GetRTDType (p. 457)

2.132.2.3 UeiDaqAPI double UeiDaq::CUeiSimulatedRTDChannel::GetRTDNominalResistance ()

Get the simulated RTD nominal resistance at 0 deg.

Returns:

The simulated RTD nominal resistance.

See also:

SetRTDNominalResistance (p. 458)

2.132.2.4 UeiDaqAPI void UeiDaq::CUeiSimulatedRTDChannel::SetRTDNominalResistance (double *r0*)

Set the simulated RTD nominal resistance at 0 deg.

Parameters:

r0 The simulated RTD nominal resistance

See also:

GetRTDNominalResistance (p. 458)

2.132.2.5 UeiDaqAPI double UeiDaq::CUeiSimulatedRTDChannel::GetCoefficientA ()

The Callendar-Van Dusen equation is an equation that describes the relationship between resistance (R) and temperature (t) of platinum resistance thermometers. $t < 0$: $R(t) = R(0)[1 + A * t + B * t * t + (t - 100)C * t * t * t]$. $t > 0$: $R(t) = R(0)(1 + A * t + B * t * t)$.

Returns:

The CVD A coefficient.

See also:

SetCoefficientA (p. 459)

2.132.2.6 UeiDaqAPI void UeiDaq::CUeiSimulatedRTDChannel::SetCoefficientA (double *A*)

The Callendar-Van Dusen equation is an equation that describes the relationship between resistance (R) and temperature (t) of platinum resistance thermometers. t < 0: R(t) = R(0)[1 + A * t + B * t * t + (t - 100)C * t * t * t]. t > 0: R(t) = R(0)(1 + A * t + B * t * t).

Parameters:

A The CVD A coefficient

See also:

GetCoefficientA (p. 458)

2.132.2.7 UeiDaqAPI double UeiDaq::CUeiSimulatedRTDChannel::GetCoefficientB ()

The Callendar-Van Dusen equation is an equation that describes the relationship between resistance (R) and temperature (t) of platinum resistance thermometers. t < 0: R(t) = R(0)[1 + A * t + B * t * t + (t - 100)C * t * t * t]. t > 0: R(t) = R(0)(1 + A * t + B * t * t).

Returns:

The CVD B coefficient.

See also:

SetCoefficientB (p. 459)

2.132.2.8 UeiDaqAPI void UeiDaq::CUeiSimulatedRTDChannel::SetCoefficientB (double *B*)

The Callendar-Van Dusen equation is an equation that describes the relationship between resistance (R) and temperature (t) of platinum resistance thermometers. t < 0: R(t) = R(0)[1 + A * t + B * t * t + (t - 100)C * t * t * t]. t > 0: R(t) = R(0)(1 + A * t + B * t * t).

Parameters:

B The CVD B coefficient

See also:

GetCoefficientB (p. 459)

2.132.2.9 UeiDaqAPI double UeiDaq::CUeiSimulatedRTDChannel::GetCoefficientC ()

The Callendar-Van Dusen equation is an equation that describes the relationship between resistance (R) and temperature (t) of platinum resistance thermometers. t < 0: R(t) = R(0)[1 + A * t + B * t * t + (t - 100)C * t * t * t]. t > 0: R(t) = R(0)(1 + A * t + B * t * t).

Returns:

The CVD C coefficient.

See also:

SetCoefficientC (p. 460)

2.132.2.10 UeiDaqAPI void UeiDaq::CUeiSimulatedRTDChannel::SetCoefficientC (double *C*)

The Callendar-Van Dusen equation is an equation that describes the relationship between resistance (R) and temperature (t) of platinum resistance thermometers. t < 0: R(t) = R(0)[1 + A * t + B * t * t + (t - 100)C * t * t * t]. t > 0: R(t) = R(0)(1 + A * t + B * t * t).

Parameters:

C The CVD C coefficient

See also:

GetCoefficientC (p. 459)

2.132.2.11 UeiDaqAPI tUeiTemperatureScale UeiDaq::CUeiSimulatedRTDChannel::GetTemperatureScale ()

Get the temperature scale used to convert the simulated temperature to resistance.

Returns:

the temperature scale.

See also:

SetTemperatureScale (p. 460)

2.132.2.12 UeiDaqAPI void UeiDaq::CUeiSimulatedRTDChannel::SetTemperatureScale (tUeiTemperatureScale *tempScale*)

Set the temperature scale used to convert the simulated temperature to resistance.

Parameters:

tempScale the temperature scale.

See also:

GetTemperatureScale (p. 460)

2.132.2.13 UeiDaqAPI bool UeiDaq::CUeiSimulatedRTDChannel::IsCircuitBreakerEnabled ()

Return true if circuit breaker for this channel is enabled and false otherwise

Returns:

Circuit breaker state

2.132.2.14 UeiDaqAPI void UeiDaq::CUeiSimulatedRTDChannel::EnableCircuitBreaker (bool *enable*)

Enable or disable circuit breaker on this channel. when enabled a circuit breaker monitors up a diagnostic channel and opens the circuit if any of the measurements goes out of pre-set limits.

Parameters:

enable True to turn-on protection, false to turn it off

2.132.2.15 UeiDaqAPI double UeiDaq::CUeiSimulatedRTDChannel::GetCircuitBreaker-CurrentLimit ()

Each circuit-breaker can monitor one diagnostic channels. Get the maximum current allowed on the specified channel. The circuit will open if more than the maximum value is monitored.

Returns:

The current limit.

2.132.2.16 UeiDaqAPI void UeiDaq::CUeiSimulatedRTDChannel::SetCircuitBreaker-CurrentLimit (double *currentLimit*)

Each circuit-breaker can monitor one diagnostic channel. Set the maximum current allowed on the specified channel. The circuit will open if more than the maximum value is monitored.

Parameters:

currentLimit The new current limit.

2.132.2.17 UeiDaqAPI double UeiDaq::CUeiSimulatedRTDChannel::GetCircuitBreaker-TemperatureLimit ()

Each circuit-breaker can monitor one diagnostic channel. Get the maximum temperature allowed on the specified channel. The circuit will open if more than the maximum value is monitored.

Returns:

The current temperature limit in celsius.

2.132.2.18 UeiDaqAPI void UeiDaq::CUeiSimulatedRTDChannel::SetCircuitBreaker-TemperatureLimit (double *temperatureLimit*)

Each circuit-breaker can monitor one diagnostic channel. Set the maximum temperature allowed on the specified channel. The circuit will open if more than the maximum value is monitored.

Parameters:

temperatureLimit The new temperature limit in celsius.

2.132.2.19 UeiDaqAPI uInt32 UeiDaq::CUeiSimulatedRTDChannel::GetOverUnderCount (void)

Specifies number of consecutive over/under limit diagnostic readings that must occur in order to trip breaker.

Returns:

The maximum number of over/under readings.

See also:

SetOverUnderCount (p. 462)

2.132.2.20 UeiDaqAPI void UeiDaq::CUeiSimulatedRTDChannel::SetOverUnderCount (uInt32 *overUnderCount*)

Specifies number of consecutive over/under limit diagnostic readings that must occur in order to trip breaker.

Parameters:

overUnderCount The new maximum number of over/under readings.

See also:

GetOverUnderCount (p. 462)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.133 UeiDaq::CUeiSimulatedSynchroResolverChannel Class Reference

Manages settings for each simulated Synchro/Resolver output channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiAOChannel**.

Public Member Functions

- UeiDaqAPI **CUeiSimulatedSynchroResolverChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiSimulatedSynchroResolverChannel** ()
  Destructor.
- UeiDaqAPI **tUeiSynchroResolverMode GetMode** ()
  Get the type of sensor simulated.
- UeiDaqAPI void **SetMode** (**tUeiSynchroResolverMode** mode)
  Set the type of sensor simulated.
- UeiDaqAPI bool **IsExternalExcitationEnabled** ()
  Get the external excitation state.
- UeiDaqAPI void **EnableExternalExcitation** (bool enabled)
  Enable or Disable external excitation.
- UeiDaqAPI double **GetExcitationVoltage** ()
  Get the excitation RMS voltage.
- UeiDaqAPI void **SetExcitationVoltage** (double vex)
  Set the excitation RMS voltage.
- UeiDaqAPI double **GetExcitationFrequency** ()
  Get the excitation frequency.
- UeiDaqAPI void **SetExcitationFrequency** (double fex)
  Set the excitation frequency.
- UeiDaqAPI bool **IsExternalAmplitudeAutoFollowEnabled** ()
  Get external amplitude auto follow.
- UeiDaqAPI void **EnableExternalAmplitudeAutoFollow** (bool enable)
  Set external amplitude auto follow.
- UeiDaqAPI int **GetPhaseDelay** ()
  Get the phase delay.

- UeiDaqAPI void **SetPhaseDelay** (int phaseDelay)
  Set the phase delay.

2.133.1 Detailed Description

Manages settings for each simulated Synchro/Resolver channel

2.133.2 Member Function Documentation

2.133.2.1 UeiDaqAPI tUeiSynchroResolverMode UeiDaq::CUeiSimulatedSynchroResolverChannel::GetMode ()

Specifies whether the sensor simulated is a synchro or a resolver.

Returns:

The current simulated sensor type.

See also:

SetMode (p. 464)

2.133.2.2 UeiDaqAPI void UeiDaq::CUeiSimulatedSynchroResolverChannel::SetMode (tUeiSynchroResolverMode *mode*)

Specifies whether the sensor simulated is a synchro or a resolver.

Parameters:

mode The new simulated sensor type.

See also:

GetMode (p. 464)

2.133.2.3 UeiDaqAPI bool UeiDaq::CUeiSimulatedSynchroResolverChannel::IsExternalExcitationEnabled ()

Determines whether the excitation will be provided by the acquisition device or by an external source

Returns:

the external excitation state.

See also:

EnableExternalExcitation (p. 465)

2.133.2.4 UeiDaqAPI void UeiDaq::CUeiSimulatedSynchroResolverChannel::EnableExternalExcitation (bool *enabled*)

Enable or disable the external excitation.

Parameters:

enabled The new external excitation state

See also:

IsExternalExcitationEnabled (p. 464)

2.133.2.5 UeiDaqAPI double UeiDaq::CUeiSimulatedSynchroResolverChannel::GetExcitationVoltage ()

Get the excitation RMS voltage configured for the channel.

Returns:

The excitation voltage.

See also:

SetExcitationVoltage (p. 465)

2.133.2.6 UeiDaqAPI void UeiDaq::CUeiSimulatedSynchroResolverChannel::SetExcitationVoltage (double *vex*)

Set the excitation RMS voltage for this channel. This sets the amplitude of the simulated synchro resolver signals when internal excitation voltage is selected

Parameters:

vex The excitation voltage

See also:

GetExcitationVoltage (p. 465)

2.133.2.7 UeiDaqAPI double UeiDaq::CUeiSimulatedSynchroResolverChannel::GetExcitationFrequency ()

Get the excitation frequency configured for the channel.

Returns:

The excitation frequency.

See also:

SetExcitationFrequency (p. 466)

2.133.2.8 UeiDaqAPI void UeiDaq::CUeiSimulatedSynchroResolverChannel::SetExcitationFrequency (double *fex*)

Set the excitation frequency for this channel. This sets the frequency of the simulated synchro resolver signals when internal excitation voltage is selected

Parameters:

fex The excitation frequency

See also:

GetExcitationFrequency (p. 465)

2.133.2.9 UeiDaqAPI bool UeiDaq::CUeiSimulatedSynchroResolverChannel::IsExternalAmplitudeAutoFollowEnabled ()

Simulated synchro/resolver signals amplitude follows external excitation amplitude by default. This setting overrides default amplitude with amplitude specified by excitation voltage setting.

Returns:

curent external amplitude setting

2.133.2.10 UeiDaqAPI void UeiDaq::CUeiSimulatedSynchroResolverChannel::EnableExternalAmplitudeAutoFollow (bool *enable*)

Simulated synchro/resolver signals amplitude follows external excitation amplitude by default. This setting overrides default amplitude with amplitude specified by excitation voltage setting.

Parameters:

enable set new external amplitude setting

2.133.2.11 UeiDaqAPI int UeiDaq::CUeiSimulatedSynchroResolverChannel::GetPhaseDelay ()

Get the phase delay. phase delay is the number of D/A conversion cycles to offset phase of the simulation waveform

Returns:

The phase delay.

See also:

SetPhaseDelay (p. 467)

2.133.2.12 UeiDaqAPI void UeiDaq::CUeiSimulatedSynchroResolverChannel::SetPhaseDelay (int *phaseDelay*)

Get the phase delay. phase delay is the number of D/A conversion cycles to offset phase of the simulation waveform

Parameters:

phaseDelay The new phase delay

See also:

GetPhaseDelay (p. 466)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.134 UeiDaq::CUeiSimulatedTCChannel Class Reference

Manages settings for each thermocouple input channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiAOChannel**.

Public Member Functions

- UeiDaqAPI **CUeiSimulatedTCChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiSimulatedTCChannel** ()
  Destructor.
- UeiDaqAPI **tUeiThermocoupleType GetThermocoupleType** ()
  Get the thermocouple type.
- UeiDaqAPI void **SetThermocoupleType** (**tUeiThermocoupleType** tcType)
  Set the thermocouple type.
- UeiDaqAPI **tUeiTemperatureScale GetTemperatureScale** ()
  Get the temperature scale.
- UeiDaqAPI void **SetTemperatureScale** (**tUeiTemperatureScale** tempScale)
  Set the temperature scale.
- UeiDaqAPI bool **IsCJCEnabled** ()
  Get the cold junction compensation state.
- UeiDaqAPI void **EnableCJC** (bool enableCJC)
  Enable or disable the cold junction compensation.
- UeiDaqAPI **UeiDaq::f64 GetCJCConstant** ()
  Get the cold junction compensation constant.
- UeiDaqAPI void **SetCJCConstant** (**UeiDaq::f64** cjcConstant)
  Set the cold junction compensation constant.

2.134.1 Detailed Description

Manages settings for each thermocouple input channel

2.134.2 Member Function Documentation

2.134.2.1 UeiDaqAPI tUeiThermocoupleType UeiDaq::CUeiSimulatedTCChannel::GetThermocoupleType ()

Get the type of the thermocouple simulated by the channel.

Returns:

the thermocouple type.

See also:

SetThermocoupleType (p. 469)

2.134.2.2 UeiDaqAPI void UeiDaq::CUeiSimulatedTCChannel::SetThermocoupleType (tUeiThermocoupleType *tcType*)

Set the type of the thermocouple simulated by the channel.

Parameters:

tcType the thermocouple type.

See also:

GetThermocoupleType (p. 468)

2.134.2.3 UeiDaqAPI tUeiTemperatureScale UeiDaq::CUeiSimulatedTCChannel::GetTemperatureScale ()

Get the temperature scale used to convert the temperature to output voltage.

Returns:

the temperature scale.

See also:

SetTemperatureScale (p. 469)

2.134.2.4 UeiDaqAPI void UeiDaq::CUeiSimulatedTCChannel::SetTemperatureScale (tUeiTemperatureScale *tempScale*)

Set the temperature scale used to convert the temperature to output voltage.

Parameters:

tempScale the temperature scale.

See also:

GetTemperatureScale (p. 469)

2.134.2.5 UeiDaqAPI bool UeiDaq::CUeiSimulatedTCChannel::IsCJCEnabled ()

Determines whether CJC is enabled

Returns:

the cold junction compensation state.

2.134.2.6 UeiDaqAPI void UeiDaq::CUeiSimulatedTCChannel::EnableCJC (bool *enableCJC*)

Enable or disable the cold junction compensation.

Parameters:

enableCJC true to enable the cold junction compensation, false otherwise.

2.134.2.7 UeiDaqAPI UeiDaq::f64 UeiDaq::CUeiSimulatedTCChannel::GetCJCConstant ()

Get the cold junction compensation temperature constant. This setting is only used when the CJC type is set to UeiCJCTypeConstant. The unit is the same as the configured temperature scale.

Returns:

the cold junction compensation constant.

See also:

SetCJCConstant (p. 470)

2.134.2.8 UeiDaqAPI void UeiDaq::CUeiSimulatedTCChannel::SetCJCConstant (UeiDaq::f64 *cjcConstant*)

Set the cold junction compensation temperature constant. This setting is only used when the CJC type is set to UeiCJCTypeConstant. The unit is the same as the configured temperature scale.

Parameters:

cjcConstant the cold junction compensation constant.

See also:

GetCJCConstant (p. 470)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.135 UeiDaq::CUeiSSIMasterPort Class Reference

Manage SSI master port settings.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Public Member Functions

- UeiDaqAPI **CUeiSSIMasterPort** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiSSIMasterPort** ()
  Destructor.
- UeiDaqAPI unsigned int **GetBps** ()
  Get port speed in bits per second.
- UeiDaqAPI void **SetBps** (unsigned int bps)
  Set port speed in bits per second.
- UeiDaqAPI unsigned int **GetWordSize** ()
  Get word size.
- UeiDaqAPI void **SetWordSize** (unsigned int wordSize)
  Set word size.
- UeiDaqAPI bool **IsTerminationResistorEnabled** ()
  Get the termination resistor state.
- UeiDaqAPI void **EnableTerminationResistor** (bool enabled)
  Enable or disable termination resistor.
- UeiDaqAPI bool **IsClockEnabled** ()
  Get the master clock state.
- UeiDaqAPI void **EnableClock** (bool enabled)
  Enable or disable master clock.
- UeiDaqAPI double **GetMinimumDataPulseWidth** ()
  Get the minimum data pulse width in microseconds.
- UeiDaqAPI void **SetMinimumDataPulseWidth** (double minWidth)
  Set the minimum data pulse width in microseconds.
- UeiDaqAPI double **GetPauseTime** ()
  Get the pause time in microseconds.
- UeiDaqAPI void **SetPauseTime** (double pauseTime)

Set the pause time in microseconds.

- UeiDaqAPI double **GetTransferTimeout** ()
  Get the transfer timeout in microseconds.
- UeiDaqAPI void **SetTransferTimeout** (double transferTimeout)
  Set the transfer timeout in microseconds.
- UeiDaqAPI double **GetBitUpdateTime** ()
  Get the bit update time in microsconds.
- UeiDaqAPI void **SetBitUpdateTime** (double bitUpdateTime)
  Set the repetition time in microseconds.
- UeiDaqAPI bool **IsTimestampingEnabled** (void)
  Get the timestamping state.
- UeiDaqAPI void **EnableTimestamping** (bool enableTimestamping)
  Set the timestamping state.
- UeiDaqAPI **f64 GetTimestampResolution** ()
  Get the timestamp resolution.
- UeiDaqAPI void **SetTimestampResolution** (**f64** resolution)
  Set the timestamp resolution.

2.135.1 Detailed Description

This objects stores SSI port settings

2.135.2 Member Function Documentation

2.135.2.1 UeiDaqAPI unsigned int UeiDaq::CUeiSSIMasterPort::GetBps ()

Get the current port speed in bits per second (12500 or 50000)

Returns:

the current port speed

2.135.2.2 UeiDaqAPI void UeiDaq::CUeiSSIMasterPort::SetBps (unsigned int *bps*)

Set the port speed in bits per second (12500 or 50000)

Parameters:

bps the new port speed

2.135.2.3 UeiDaqAPI unsigned int UeiDaq::CUeiSSIMasterPort::GetWordSize ()

Get the current word size (3 to 32 bits)

Returns:

the current word size

2.135.2.4 UeiDaqAPI void UeiDaq::CUeiSSIMasterPort::SetWordSize (unsigned int *wordSize*)

Set the word size (3 to 32 bits)

Parameters:

wordSize the new word size

2.135.2.5 UeiDaqAPI bool UeiDaq::CUeiSSIMasterPort::IsTerminationResistorEnabled ()

Determines whether the termination resistor is enabled

Returns:

The termination resistor state.

See also:

EnableTerminationResistor (p. 473)

2.135.2.6 UeiDaqAPI void UeiDaq::CUeiSSIMasterPort::EnableTerminationResistor (bool *enabled*)

Enable or disable the termination resistor.

Parameters:

enabled The new termination resistor state

See also:

IsTerminationResistorEnabled (p. 473)

2.135.2.7 UeiDaqAPI bool UeiDaq::CUeiSSIMasterPort::IsClockEnabled ()

Determines whether the master is emitting the clock signal

Returns:

The master clock state.

See also:

EnableClock (p. 474)

2.135.2.8 UeiDaqAPI void UeiDaq::CUeiSSIMasterPort::EnableClock (bool *enabled*)

Enable or disable the master clock output.

Parameters:

enabled The new master clock state

See also:

IsClockEnabled (p. 473)

2.135.2.9 UeiDaqAPI double UeiDaq::CUeiSSIMasterPort::GetMinimumDataPulseWidth ()

Get the minimum pulse width in us the master recognizes at its data input. Any pulse whose width is smaller will be ignored.

Returns:

the minimum pulse width

See also:

SetMinimumDataPulseWidth (p. 474)

2.135.2.10 UeiDaqAPI void UeiDaq::CUeiSSIMasterPort::SetMinimumDataPulseWidth (double *minWidth*)

Set the minimum pulse width in us the master recognizes at its data input. Any pulse whose width is smaller will be ignored. Set the minimum pulse width to 0 to disable digital filtering.

Parameters:

minWidth the minimum data pulse width

See also:

GetMinimumDataPulseWidth (p. 474)

2.135.2.11 UeiDaqAPI double UeiDaq::CUeiSSIMasterPort::GetPauseTime ()

Get the time delay between two consecutive clock sequences from the master. Pause time is typically named Tp

Returns:

the pause time

See also:

SetPauseTime (p. 475)

2.135.2.12 UeiDaqAPI void UeiDaq::CUeiSSIMasterPort::SetPauseTime (double *pauseTime*)

Set the time delay between two consecutive clock sequences from the master. Pause time is typically named Tp

Parameters:

pauseTime the new pause time

See also:

GetPauseTime (p. 474)

2.135.2.13 UeiDaqAPI double UeiDaq::CUeiSSIMasterPort::GetTransferTimeout ()

Get the minimum time required by the slave to realise that the data transmission is complete Transfer timeout is typically named Tm

Returns:

the transfer timeout

See also:

SetTransferTimeout (p. 475)

2.135.2.14 UeiDaqAPI void UeiDaq::CUeiSSIMasterPort::SetTransferTimeout (double *transferTimeout*)

Set the minimum time required by the slave to realise that the data transmission is complete Transfer timeout is typically named Tm

Parameters:

transferTimeout the new transfer timeout

See also:

GetTransferTimeout (p. 475)

2.135.2.15 UeiDaqAPI double UeiDaq::CUeiSSIMasterPort::GetBitUpdateTime ()

Get the maximum duration for a low to high transition of high to low transition Bit update time is typically named Tv

Returns:

the bit update time

See also:

SetBitupdateTime

2.135.2.16 UeiDaqAPI void UeiDaq::CUeiSSIMasterPort::SetBitUpdateTime (double *bitUpdateTime*)

Set the maximum duration for a low to high transition of high to low transition Bit update time is typically named Tv

Parameters:

bitUpdateTime the new bit update time

See also:

GetBitUpdateTime (p. 475)

2.135.2.17 UeiDaqAPI bool UeiDaq::CUeiSSIMasterPort::IsTimestampingEnabled (void)

Determines whether each received frame is timestamped.

Returns:

The timestamping state.

See also:

EnableTimestamping (p. 476)

2.135.2.18 UeiDaqAPI void UeiDaq::CUeiSSIMasterPort::EnableTimestamping (bool *enableTimestamping*)

Specifies whether each received frame is timestamped.

Parameters:

enableTimestamping true to turn timestamping on, false otherwise

See also:

IsTimestampingEnabled (p. 476)

2.135.2.19 UeiDaqAPI f64 UeiDaq::CUeiSSIMasterPort::GetTimestampResolution ()

Get the timestamp resolution in seconds.

Returns:

the timestamp resolution.

See also:

SetTimestampResolution (p. 477)

2.135.2.20 UeiDaqAPI void UeiDaq::CUeiSSIMasterPort::SetTimestampResolution (f64 *resolution*)

Set the timestamp resolution in seconds.

Parameters:

resolution the new resolution.

See also:

GetTimestampResolution (p. 476)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.136 UeiDaq::CUeiSSIReader Class Reference

SSI Reader class.

`#include <UeiMessaging.h>`

Public Member Functions

- UeiDaqAPI **CUeiSSIReader** (**CUeiDataStream** *pDataStream, **Int32** port=0)
  Constructor.
- virtual UeiDaqAPI ~**CUeiSSIReader** ()
  Destructor.
- UeiDaqAPI void **Read** (bool grayDecoding, **Int32** numWords, **uInt32** *pBuffer, **Int32** *numWordsRead)
  Read word(s) from the SSI master port.

2.136.1 Detailed Description

This class handles reading data words of a stream coming from an SSI master interface. It returns 32-bit data word(s).

2.136.2 Constructor & Destructor Documentation

2.136.2.1 UeiDaqAPI UeiDaq::CUeiSSIReader::CUeiSSIReader (CUeiDataStream * *pDataStream*, Int32 *port* = 0)

Parameters:

pDataStream represents the source of data to read

port the SSI master port

2.136.3 Member Function Documentation

2.136.3.1 UeiDaqAPI void UeiDaq::CUeiSSIReader::Read (bool *grayDecoding*, Int32 *numWords*, uInt32 * *pBuffer*, Int32 * *numWordsRead*)

Read 32-bit data words from the input stream

Parameters:

grayDecoding true to convert gray encoded values to binary

numWords the number of words that can be stored in the destination buffer

pBuffer destination buffer

numWordsRead the actual number of frames read from the SSI port

The documentation for this class was generated from the following file:

- UeiMessaging.h

2.137 UeiDaq::CUeiSSISlavePort Class Reference

Manage SSI slave port settings.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Public Member Functions

- UeiDaqAPI **CUeiSSISlavePort** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiSSISlavePort** ()
  Destructor.
- UeiDaqAPI unsigned int **GetBps** ()
  Get port speed in bits per second.
- UeiDaqAPI void **SetBps** (unsigned int bps)
  Set port speed in bits per second.
- UeiDaqAPI unsigned int **GetWordSize** ()
  Get word size.
- UeiDaqAPI void **SetWordSize** (unsigned int wordSize)
  Set word size.
- UeiDaqAPI bool **IsTerminationResistorEnabled** ()
  Get the termination resistor state.
- UeiDaqAPI void **EnableTerminationResistor** (bool enabled)
  Enable or disable termination resistor.
- UeiDaqAPI bool **IsTransmitEnabled** ()
  Get the slave transmit state.
- UeiDaqAPI void **EnableTransmit** (bool enabled)
  Enable or disable data transmission.
- UeiDaqAPI double **GetMinimumClockPulseWidth** ()
  Get the minimum clock pulse width in microseconds.
- UeiDaqAPI void **SetMinimumClockPulseWidth** (double minWidth)
  Set the minimum clock pulse width in microseconds.
- UeiDaqAPI double **GetPauseTime** ()
  Get the pause time in microseconds.
- UeiDaqAPI void **SetPauseTime** (double pauseTime)

Set the pause time in microseconds.

- UeiDaqAPI double **GetTransferTimeout** ()

  Get the transfer timeout in microseconds.

- UeiDaqAPI void **SetTransferTimeout** (double transferTimeout)

  Set the transfer timeout in microseconds.

- UeiDaqAPI double **GetBitUpdateTime** ()

  Get the bit update time in microsconds.

- UeiDaqAPI void **SetBitUpdateTime** (double bitUpdateTime)

  Set the repetition time in microseconds.

2.137.1 Detailed Description

This objects stores SSI port settings

2.137.2 Member Function Documentation

2.137.2.1 UeiDaqAPI unsigned int UeiDaq::CUeiSSISlavePort::GetBps ()

Get the current port speed in bits per second (12500 or 50000)

Returns:

the current port speed

2.137.2.2 UeiDaqAPI void UeiDaq::CUeiSSISlavePort::SetBps (unsigned int *bps*)

Set the port speed in bits per second (12500 or 50000)

Parameters:

bps the new port speed

2.137.2.3 UeiDaqAPI unsigned int UeiDaq::CUeiSSISlavePort::GetWordSize ()

Get the current word size (3 to 32 bits)

Returns:

the current word size

2.137.2.4 UeiDaqAPI void UeiDaq::CUeiSSISlavePort::SetWordSize (unsigned int *wordSize*)

Set the word size (3 to 32 bits)

Parameters:

wordSize the new word size

2.137.2.5 UeiDaqAPI bool UeiDaq::CUeiSSISlavePort::IsTerminationResistorEnabled ()

Determines whether the termination resistor is enabled

Returns:

The termination resistor state.

See also:

EnableTerminationResistor (p. 482)

2.137.2.6 UeiDaqAPI void UeiDaq::CUeiSSISlavePort::EnableTerminationResistor (bool *enabled*)

Enable or disable the termination resistor.

Parameters:

enabled The new termination resistor state

See also:

IsTerminationResistorEnabled (p. 482)

2.137.2.7 UeiDaqAPI bool UeiDaq::CUeiSSISlavePort::IsTransmitEnabled ()

Determines whether the master will transmit data upon receiving the clock signal

Returns:

The slave transmit state.

See also:

EnableTransmit (p. 482)

2.137.2.8 UeiDaqAPI void UeiDaq::CUeiSSISlavePort::EnableTransmit (bool *enabled*)

Enable or disable the data output.

Parameters:

enabled The new master clock state

See also:

IsTransmitEnabled (p. 482)

2.137.2.9 UeiDaqAPI double UeiDaq::CUeiSSISlavePort::GetMinimumClockPulseWidth ()

Get the minimum pulse width in us the slave recognizes at its clock input. Any pulse whose width is smaller will be ignored.

Returns:

the minimum pulse width

See also:

SetMinimumClockPulseWidth (p. 483)

2.137.2.10 UeiDaqAPI void UeiDaq::CUeiSSISlavePort::SetMinimumClockPulseWidth (double *minWidth*)

Set the minimum pulse width in us the slave recognizes at its clock input. Any pulse whose width is smaller will be ignored. Set the minimum pulse width to 0 to disable digital filtering.

Parameters:

minWidth the minimum clock pulse width

See also:

GetMinimumClockPulseWidth (p. 483)

2.137.2.11 UeiDaqAPI double UeiDaq::CUeiSSISlavePort::GetPauseTime ()

Get the time delay between two consecutive clock sequences from the master. Pause time is typically named Tp

Returns:

the pause time

See also:

SetPauseTime (p. 483)

2.137.2.12 UeiDaqAPI void UeiDaq::CUeiSSISlavePort::SetPauseTime (double *pauseTime*)

Set the time delay between two consecutive clock sequences from the master. Pause time is typically named Tp

Parameters:

pauseTime the new pause time

See also:

GetPauseTime (p. 483)

2.137.2.13 UeiDaqAPI double UeiDaq::CUeiSSISlavePort::GetTransferTimeout ()

Get the minimum time required by the slave to realise that the data transmission is complete Transfer timeout is typically named Tm

Returns:

the transfer timeout

See also:

SetTransferTimeout (p. 484)

2.137.2.14 UeiDaqAPI void UeiDaq::CUeiSSISlavePort::SetTransferTimeout (double *transferTimeout*)

Set the minimum time required by the slave to realise that the data transmission is complete Transfer timeout is typically named Tm

Parameters:

transferTimeout the new transfer timeout

See also:

GetTransferTimeout (p. 484)

2.137.2.15 UeiDaqAPI double UeiDaq::CUeiSSISlavePort::GetBitUpdateTime ()

Get the maximum duration for a low to high transition of high to low transition Bit update time is typically named Tv

Returns:

the bit update time

See also:

SetBitupdateTime

2.137.2.16 UeiDaqAPI void UeiDaq::CUeiSSISlavePort::SetBitUpdateTime (double *bitUpdateTime*)

Set the maximum duration for a low to high transition of high to low transition Bit update time is typically named Tv

Parameters:

bitUpdateTime the new bit update time

See also:

GetBitUpdateTime (p. 484)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.138 UeiDaq::CUeiSSIWriter Class Reference

SSI Writer class.

`#include <UeiMessaging.h>`

Public Member Functions

- UeiDaqAPI **CUeiSSIWriter** (**CUeiDataStream** *pDataStream, **Int32** port=0)
  Constructor.
- virtual UeiDaqAPI ~**CUeiSSIWriter** ()
  Destructor.
- UeiDaqAPI void **Write** (bool grayEncoding, **Int32** numWords, **uInt32** *pBuffer, **Int32** *numWordsWritten)
  Write word(s) to the SSI slave port.

2.138.1 Detailed Description

This class handles writing data words of a stream associated with an SSI slave interface. It write 32-bit data word(s).

2.138.2 Constructor & Destructor Documentation

2.138.2.1 UeiDaqAPI UeiDaq::CUeiSSIWriter::CUeiSSIWriter (CUeiDataStream * *pDataStream*, Int32 *port* = 0)

Parameters:

pDataStream represents the destination where data is written to

port the SSI slave port

2.138.3 Member Function Documentation

2.138.3.1 UeiDaqAPI void UeiDaq::CUeiSSIWriter::Write (bool *grayEncoding*, Int32 *numWords*, uInt32 * *pBuffer*, Int32 * *numWordsWritten*)

Write 32-bit data words to the output stream

Parameters:

grayEncoding true to convert binary values to gray

numWords the number of words to write

pBuffer source buffer

numWordsWritten the actual number of frames sent to the SSI port

The documentation for this class was generated from the following file:

- UeiMessaging.h

2.139 UeiDaq::CUeiSync1PPSController Class Reference

1PPS synchronization controller class

`#include <UeiMessaging.h>`

Public Member Functions

- UeiDaqAPI **CUeiSync1PPSController** (**CUeiDataStream** *pDataStream, **Int32** channel=0)

  Constructor.

- virtual UeiDaqAPI ~**CUeiSync1PPSController** ()

  Destructor.

- UeiDaqAPI void **ReadStatus** (**tUeiSync1PPSStatus** *pBuffer)

  Read latest status.

- UeiDaqAPI void **ReadLockedStatus** (bool *isLocked)

  Read event module status.

- UeiDaqAPI void **ReadPTPStatus** (**tUeiSync1PPSPTPStatus** *pBuffer)

  Read PTP status.

- UeiDaqAPI void **ReadPTPUTCTime** (**tUeiPTPTime** *pBuffer)

  Read PTP UTC time.

- UeiDaqAPI void **TriggerDevices** (**tUeiSync1PPSTriggerOperation** trigOp, bool resetTimestamp)

  Trigger slave devices.

2.139.1 Detailed Description

Class that handles reading 1PPS status and sending trigger messages via a stream associated with a sync capable CPU device

2.139.2 Constructor & Destructor Documentation

2.139.2.1 UeiDaqAPI UeiDaq::CUeiSync1PPSController::CUeiSync1PPSController (CUeiDataStream * *pDataStream*, Int32 *channel* = 0)

Parameters:

pDataStream represents the source where to read data

channel Unused

2.139.3 Member Function Documentation

2.139.3.1 UeiDaqAPI void UeiDaq::CUeiSync1PPSController::ReadStatus (tUeiSync1PPSStatus * *pBuffer*)

Read latest status.

Parameters:

pBuffer destination buffer

2.139.3.2 UeiDaqAPI void UeiDaq::CUeiSync1PPSController::ReadLockedStatus (bool * *isLocked*)

Event module status is true when the event module output clock is stable enough

Parameters:

isLocked true if the event module output is stable, false otherwise

2.139.3.3 UeiDaqAPI void UeiDaq::CUeiSync1PPSController::ReadPTPStatus (tUeiSync1PPSPTPStatus * *pBuffer*)

Read latest PTP status

Parameters:

pBuffer destination buffer

2.139.3.4 UeiDaqAPI void UeiDaq::CUeiSync1PPSController::ReadPTPUTCTime (tUeiPTPTime * *pBuffer*)

Read current UTC time from PTP clock

Parameters:

pBuffer destination buffer

2.139.3.5 UeiDaqAPI void UeiDaq::CUeiSync1PPSController::TriggerDevices (tUeiSync1PPSTriggerOperation *trigOp*, bool *resetTimestamp*)

Trigger devices configured to use clock signal generated by the 1PPs circuitry

Parameters:

trigOp specify the method used to send the trigger signal to all slave devices

resetTimestamp True if all slave devices should reset their timestamp counter upon receiving the trigger signal

The documentation for this class was generated from the following file:

- UeiMessaging.h

2.140 UeiDaq::CUeiSync1PPSPort Class Reference

Manage 1PPS synchronization settings.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Public Member Functions

- UeiDaqAPI **CUeiSync1PPSPort** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiSync1PPSPort** ()
  Destructor.
- UeiDaqAPI **tUeiSync1PPSMode Get1PPSMode** ()
  Get the current 1PPS sync mode.
- UeiDaqAPI void **Set1PPSMode** (**tUeiSync1PPSMode** mode)
  Set the 1PPS sync mode.
- UeiDaqAPI **tUeiSync1PPSSource Get1PPSSource** ()
  Get the current 1PPS clock source.
- UeiDaqAPI void **Set1PPSSource** (**tUeiSync1PPSSource** source)
  Set the 1PPS clock source.
- UeiDaqAPI **tUeiSync1PPSOutput Get1PPSOutput** ()
  Get the current 1PPS output.
- UeiDaqAPI void **Set1PPSOutput** (**tUeiSync1PPSOutput** output)
  Set the 1PPS output.
- UeiDaqAPI **tUeiSyncLine Get1PPSToADPLLSyncLine** ()
  Get the sync line connecting the 1PPS clock to the ADPLL.
- UeiDaqAPI void **Set1PPSToADPLLSyncLine** (**tUeiSyncLine** line)
  Set the sync line connecting the 1PPS clock to the ADPLL.
- UeiDaqAPI int **GetNumPPS** ()
  Get the number of pulses per second.
- UeiDaqAPI void **SetNumPPS** (int pps)
  Set the number of pulses per second.
- UeiDaqAPI int **Get1PPSAccuracy** ()
  Get the 1PPS accuracy.
- UeiDaqAPI void **Set1PPSAccuracy** (int accuracy)

Set the 1PPS accuracy.

- UeiDaqAPI **tUeiSyncLine GetADPLLOutputSyncLine** ()
  Get the sync line connected to the ADPLL output.
- UeiDaqAPI void **SetADPLLOutputSyncLine** (**tUeiSyncLine** line)
  Set the sync line connected to the ADPLL output.
- UeiDaqAPI **tUeiSyncLine GetEMOutputSyncLine** ()
  Get the sync line connecting the event module output to the I/O layers.
- UeiDaqAPI void **SetEMOutputSyncLine** (**tUeiSyncLine** line)
  Set the sync line connecting the event module output to the I/O layers.
- UeiDaqAPI double **GetEMOutputRate** ()
  Get the event module output clock rate.
- UeiDaqAPI void **SetEMOutputRate** (double rate)
  Set the event module output clock rate.
- UeiDaqAPI **tUeiSyncLine GetTriggerOutputSyncLine** ()
  Get the sync line connecting the 1PPS trigger to the I/O layers.
- UeiDaqAPI void **SetTriggerOutputSyncLine** (**tUeiSyncLine** line)
  Set the sync line connecting the 1PPS trigger to the I/O layers.
- UeiDaqAPI int **GetPTPEthernetPort** ()
  Get the PTP ethernet port.
- UeiDaqAPI void **SetPTPEthernetPort** (int ethernetPort)
  Set the PTP ethernet port.
- UeiDaqAPI **uInt8 GetPTPSubdomain** ()
  Get the PTP subdomain.
- UeiDaqAPI void **SetPTPSubdomain** (**uInt8** subDomain)
  Set the PTP subdomain.
- UeiDaqAPI **uInt8 GetPTPPriority1** ()
  Get the first order PTP priority.
- UeiDaqAPI void **SetPTPPriority1** (**uInt8** priority1)
  Set the first order PTP priority.
- UeiDaqAPI **uInt8 GetPTPPriority2** ()
  Get the second order PTP priority.
- UeiDaqAPI void **SetPTPPriority2** (**uInt8** priority2)
  Set the second order PTP priority.

- UeiDaqAPI **Int8 GetPTPLogSyncInterval** ()
  Get the sync interval.
- UeiDaqAPI void **SetPTPLogSyncInterval** (**Int8** logSyncInterval)
  Set the sync interval.
- UeiDaqAPI **Int8 GetPTPLogMinDelayRequestInterval** ()
  Get the minimum delay request interval.
- UeiDaqAPI void **SetPTPLogMinDelayRequestInterval** (**Int8** logMinDelayRequest-Interval)
  Set the minimum delay request interval.
- UeiDaqAPI **Int8 GetPTPLogAnnounceInterval** ()
  Get the interval between PTP announce messages.
- UeiDaqAPI void **SetPTPLogAnnounceInterval** (**Int8** logAnnouceInterval)
  Set the interval between PTP announce messages.
- UeiDaqAPI **uInt8 GetPTPAnnounceTimeout** ()
  Get the number of PTP intervals before a timeout occurs.
- UeiDaqAPI void **SetPTPAnnounceTimeout** (**uInt8** announceTimeout)
  Set the number of PTP intervals before a timeout occurs.
- UeiDaqAPI **uInt8 GetPTPUTCOffset** ()
  Get the UTC offset.
- UeiDaqAPI void **SetPTPUTCOffset** (**uInt8** utcOffset)
  Set the UTC offset.
- UeiDaqAPI **uInt32 GetSyncInputDebounceTime** ()
  Get the debounce time for sync input.
- UeiDaqAPI void **SetSyncInputDebounceTime** (**uInt32** debounceTimeNs)
  Set the debounce time for sync input.

2.140.1 Detailed Description

This objects stores 1PPS synchronization settings The 1PPS synchronization uses an ADPLL and an event module to produce a clock at an arbitrary rate Multiple racks can easily be synchronized when they use the same 1PPS synchronization clock The ADPLL+event module on each rack will produce synchronized scan clocks

The source of the 1PPS sync clock can be internal, external or generated from NTP The 1PPS clock is routed via one of the four internal sync lines to the ADPLL (adaptive digital PLL) The ADPLL locks on the 1PPS and outputs its own 1PPS that is an average of the original 1PPS clock The ADPLL can maintain its 1PPS output even if the original 1PPS clock gets disconnected The ADPLL 1PPS output is routed via one of the four sync lines to the event module The ADPLL

1PPS output can also be routed to the sync connector for sharing with other racks/cubes The event module produces a user selectable number of pulses upon every 1PPS pulse coming from the ADPLL. The event module clock output is routed to the Input layers via one of the remaining sync lines

2.140.2 Member Function Documentation

2.140.2.1 UeiDaqAPI tUeiSync1PPSMode UeiDaq::CUeiSync1PPSPort::Get1PPSMode ()

Get the current 1PPS sync mode Clock uses a 1PPS signal produced internally or received from an external source NTP derives the 1PPS from an NTP server

Returns:

the current 1PPS mode

2.140.2.2 UeiDaqAPI void UeiDaq::CUeiSync1PPSPort::Set1PPSMode (tUeiSync1PPSMode *mode*)

Set the 1PPS sync mode Clock uses a 1PPS signal produced internally or received from an external source NTP derives the 1PPS from an NTP server

Parameters:

mode the new 1PPS mode

2.140.2.3 UeiDaqAPI tUeiSync1PPSSource UeiDaq::CUeiSync1PPSPort::Get1PPSSource ()

Get the source of the 1PPS clock in SyncClock mode: Internal when 1PPS is generated internally Input0 when external 1PPS is connected to input 0 of sync connector (trigger input) Input1 when external 1PPS is connected to input 1 of sync connector (clock input)

Returns:

the current source of the 1PPS

2.140.2.4 UeiDaqAPI void UeiDaq::CUeiSync1PPSPort::Set1PPSSource (tUeiSync1PPSSource *source*)

Set the source of the 1PPS clock in SyncClock mode: Internal when 1PPS is generated internally Input0 when external 1PPS is connected to input 0 of sync connector (trigger input) Input1 when external 1PPS is connected to input 1 of sync connector (clock input)

Parameters:

source the new 1PPS source

2.140.2.5 UeiDaqAPI tUeiSync1PPSOutput UeiDaq::CUeiSync1PPSPort::Get1PPSOutput ()

When module is a master (1PPS clock source is internal) Get the pin used to output the 1PPS sync clock Output0 when 1PPS is emitted out of output 0 of sync connector (trigger output) Output1 when 1PPS is emitted out of output 1 of sync connector (clock output)

Returns:

the current 1PPS output

2.140.2.6 UeiDaqAPI void UeiDaq::CUeiSync1PPSPort::Set1PPSOutput (tUeiSync1PPSOutput *output*)

When module is a master (1PPS clock source is internal) Set the pin used to output the 1PPS sync clock Output0 when 1PPS is emitted out of output 0 of sync connector (trigger output) Output1 when 1PPS is emitted out of output 1 of sync connector (clock output)

Parameters:

output the new 1PPS output

2.140.2.7 UeiDaqAPI tUeiSyncLine UeiDaq::CUeiSync1PPSPort::Get1PPSToADPLLSyncLine ()

Get the internal sync line connecting the internal or external 1PPS clock to the ADPLL

Returns:

the current sync line connecting 1PPS to ADPLL

2.140.2.8 UeiDaqAPI void UeiDaq::CUeiSync1PPSPort::Set1PPSToADPLLSyncLine (tUeiSyncLine *line*)

Set the internal sync line connecting the internal or external 1PPS clock to the ADPLL

Parameters:

line the new sync line used to connect 1PPS to ADPLL

2.140.2.9 UeiDaqAPI int UeiDaq::CUeiSync1PPSPort::GetNumPPS ()

Get the number of pulses per second used by the synchronization clock signal Default is 1

Returns:

the number of pulses per second

2.140.2.10 UeiDaqAPI void UeiDaq::CUeiSync1PPSPort::SetNumPPS (int *pps*)

Set the number of pulses per second used by the synchronization clock signal Default is 1

Parameters:

pps the new number of pulses per second

2.140.2.11 UeiDaqAPI int UeiDaq::CUeiSync1PPSPort::Get1PPSAccuracy ()

Get the required accuracy in microsecs of the 1PPS clock signal. Clocks that are out of range are ignored

Returns:

the 1PPS minimal accuracy

2.140.2.12 UeiDaqAPI void UeiDaq::CUeiSync1PPSPort::Set1PPSAccuracy (int *accuracy*)

Set the expected accuracy in microsecs of the 1PPS clock signal. Clocks that are out of range are ignored

Parameters:

accuracy the new 1PPS minimal accuracy

2.140.2.13 UeiDaqAPI tUeiSyncLine UeiDaq::CUeiSync1PPSPort::GetADPLLOutputSyncLine ()

Get the internal sync line connected to the ADPLL output

Returns:

the sync line used to connect the ADPLL output

2.140.2.14 UeiDaqAPI void UeiDaq::CUeiSync1PPSPort::SetADPLLOutputSyncLine (tUeiSyncLine *line*)

Set the internal sync line connected to the ADPLL output

Parameters:

line the new sync line used to connect the ADPLL output

2.140.2.15 UeiDaqAPI tUeiSyncLine UeiDaq::CUeiSync1PPSPort::GetEMOutputSyncLine ()

Get the internal sync line connecting the event module output to the I/O layers The event module produces the clock used by the I/O layers to pace data acquisition or generation

Returns:

the sync line used to connect the event output module to I/O layers

2.140.2.16 UeiDaqAPI void UeiDaq::CUeiSync1PPSPort::SetEMOutputSyncLine (tUeiSyncLine *line*)

Set the internal sync line connecting the event module output to the I/O layers The event module produces the clock used by the I/O layers to pace data acquisition or generation

Parameters:

line the new sync line used to connect the event output module to I/O layers

2.140.2.17 UeiDaqAPI double UeiDaq::CUeiSync1PPSPort::GetEMOutputRate ()

Get the rate (Hz) of the clock produced by the event module

Returns:

the current event module output clock rate

2.140.2.18 UeiDaqAPI void UeiDaq::CUeiSync1PPSPort::SetEMOutputRate (double *rate*)

Set the rate (Hz) of the clock produced by the event module

Parameters:

rate the new event module output clock rate

2.140.2.19 UeiDaqAPI tUeiSyncLine UeiDaq::CUeiSync1PPSPort::GetTriggerOutputSyncLine ()

The ADPLL is capable of emitting a trigger signal simultaneously with the next 1PPS pulse Get the sync line connecting the 1PPS trigger to the I/O layers

Returns:

the sync line used to connect the 1PPS trigger to I/O layers

2.140.2.20 UeiDaqAPI void UeiDaq::CUeiSync1PPSPort::SetTriggerOutputSyncLine (tUeiSyncLine *line*)

The ADPLL is capable of emitting a trigger signal simultaneously with the next 1PPS pulse Get the sync line connecting the 1PPS trigger to the I/O layers

Parameters:

line the new sync line used to connect the 1PPS trigger to I/O layers

2.140.2.21 UeiDaqAPI int UeiDaq::CUeiSync1PPSPort::GetPTPEthernetPort ()

Get the PTP ethernet port used to connect with master clock

Returns:

the current PTP ethernet port

2.140.2.22 UeiDaqAPI void UeiDaq::CUeiSync1PPSPort::SetPTPEthernetPort (int *ethernetPort*)

Set the PTP ethernet port used to connect with master clock

Parameters:

ethernetPort the new PTP ethernet port

2.140.2.23 UeiDaqAPI uInt8 UeiDaq::CUeiSync1PPSPort::GetPTPSubdomain ()

Get the PTP subdomain which is a logical group of PTP clocks

Returns:

the current PTP domain

2.140.2.24 UeiDaqAPI void UeiDaq::CUeiSync1PPSPort::SetPTPSubdomain (uInt8 *subDomain*)

Set the PTP subdomain which is a logical group of PTP clocks

Parameters:

subDomain the new PTP domain

2.140.2.25 UeiDaqAPI uInt8 UeiDaq::CUeiSync1PPSPort::GetPTPPriority1 ()

Get the first order PTP priority used as a factor for selecting master clock or becoming master if none with higher priority is found

Returns:

the current first order PTP priority

2.140.2.26 UeiDaqAPI void UeiDaq::CUeiSync1PPSPort::SetPTPPriority1 (uInt8 *priority1*)

Set the first order PTP priority used as a factor for selecting master clock or becoming master if none with higher priority is found

Parameters:

priority1 the new first order PTP priority

2.140.2.27 UeiDaqAPI uInt8 UeiDaq::CUeiSync1PPSPort::GetPTPPriority2 ()

Get the second order PTP priority used as a factor for selecting master clock or becoming master if none with higher priority is found

Returns:

the current second order PTP priority

2.140.2.28 UeiDaqAPI void UeiDaq::CUeiSync1PPSPort::SetPTPPriority2 (uInt8 *priority2*)

Set the second order PTP priority used as a factor for selecting master clock or becoming master if none with higher priority is found

Parameters:

priority2 the new second order PTP priority

2.140.2.29 UeiDaqAPI Int8 UeiDaq::CUeiSync1PPSPort::GetPTPLogSyncInterval ()

Get the sync interval which is how often the master sends sync packets Value is specifed as the log of actual sync interval.

Returns:

the current Sync interval

2.140.2.30 UeiDaqAPI void UeiDaq::CUeiSync1PPSPort::SetPTPLogSyncInterval (Int8 *logSyncInterval*)

Set the sync interval which is how often the master sends sync packets Value is specifed as the log of actual sync interval.

Parameters:

logSyncInterval the new Sync interval

2.140.2.31 UeiDaqAPI Int8 UeiDaq::CUeiSync1PPSPort::GetPTPLogMinDelayRequest-Interval ()

Get the minimum interval allowed between PTP delay request packets Value is specifed as the log of actual interval.

Returns:

the current minimum delay request interval

2.140.2.32 UeiDaqAPI void UeiDaq::CUeiSync1PPSPort::SetPTPLogMinDelayRequestInterval (Int8 *logMinDelayRequestInterval*)

Set the minimum interval allowed between PTP delay request packets Value is specifed as the log of actual interval.

Parameters:

logMinDelayRequestInterval the new minimum delay request interval

2.140.2.33 UeiDaqAPI Int8 UeiDaq::CUeiSync1PPSPort::GetPTPLogAnnounceInterval ()

Get the the interval between PTP announce messages Value is specifed as the log of actual interval.

Returns:

the current interval between PTP announce messages

2.140.2.34 UeiDaqAPI void UeiDaq::CUeiSync1PPSPort::SetPTPLogAnnounceInterval (Int8 *logAnnouceInterval*)

Set the interval between PTP announce messages Value is specifed as the log of actual interval.

Parameters:

logAnnouceInterval the new interval between PTP announce messages

2.140.2.35 UeiDaqAPI uInt8 UeiDaq::CUeiSync1PPSPort::GetPTPAnnounceTimeout ()

Get the number of PTP intervals before a timeout occurs

Returns:

the current number of PTP intervals before a timeout occurs

2.140.2.36 UeiDaqAPI void UeiDaq::CUeiSync1PPSPort::SetPTPAnnounceTimeout (uInt8 *announceTimeout*)

Set the number of PTP intervals before a timeout occurs

Parameters:

announceTimeout the new number of PTP intervals before a timeout occurs

2.140.2.37 UeiDaqAPI uInt8 UeiDaq::CUeiSync1PPSPort::GetPTPUTCOffset ()

Get the offset between UTC and TAI in seconds

Returns:

the current UTC offset

2.140.2.38 UeiDaqAPI void UeiDaq::CUeiSync1PPSPort::SetPTPUTCOffset (uInt8 *utcOffset*)

Set the offset between UTC and TAI in seconds

Parameters:

utcOffset the new UTC offset

2.140.2.39 UeiDaqAPI uInt32 UeiDaq::CUeiSync1PPSPort::GetSyncInputDebounceTime ()

Get the amount of time during which the sync input doesn't register any incoming edge

Returns:

the current debounce time in nanoseconds

2.140.2.40 UeiDaqAPI void UeiDaq::CUeiSync1PPSPort::SetSyncInputDebounceTime (uInt32 *debounceTimeNs*)

Set the amount of time during chich the sync input doesn't register any incoming edge

Parameters:

debounceTimeNs New debounce time in nanoseconds

The documentation for this class was generated from the following file:

- UeiChannel.h

2.141 UeiDaq::CUeiSynchroResolverChannel Class Reference

Manages settings for each Synchro/Resolver input channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiAIChannel**.

Public Member Functions

- UeiDaqAPI **CUeiSynchroResolverChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiSynchroResolverChannel** ()
  Destructor.
- UeiDaqAPI **tUeiSynchroResolverMode GetMode** ()
  Get the type of input sensor.
- UeiDaqAPI void **SetMode** (**tUeiSynchroResolverMode** mode)
  Set the type of input sensor.
- UeiDaqAPI bool **IsExternalExcitationEnabled** ()
  Get the external excitation state.
- UeiDaqAPI void **EnableExternalExcitation** (bool enabled)
  Enable or Disable external excitation.
- UeiDaqAPI double **GetExcitationVoltage** ()
  Get the excitation RMS voltage.
- UeiDaqAPI void **SetExcitationVoltage** (double vex)
  Set the excitation RMS voltage.
- UeiDaqAPI double **GetExcitationFrequency** ()
  Get the excitation frequency.
- UeiDaqAPI void **SetExcitationFrequency** (double fex)
  Set the excitation frequency.

2.141.1 Detailed Description

Manages settings for each Synchro/Resolver channel

2.141.2 Member Function Documentation

2.141.2.1 UeiDaqAPI tUeiSynchroResolverMode UeiDaq::CUeiSynchroResolverChannel::GetMode ()

Specifies whether the input sensor is a synchro or a resolver.

Returns:

The current input sensor type.

See also:

SetMode (p. 502)

2.141.2.2 UeiDaqAPI void UeiDaq::CUeiSynchroResolverChannel::SetMode (tUeiSynchroResolverMode *mode*)

Specifies whether the input sensor is a synchro or a resolver.

Parameters:

mode The new input sensor type.

See also:

GetMode (p. 501)

2.141.2.3 UeiDaqAPI bool UeiDaq::CUeiSynchroResolverChannel::IsExternalExcitationEnabled ()

Determines whether the excitation will be provided by the acquisition device or by an external source

Returns:

the external excitation state.

See also:

EnableExternalExcitation (p. 502)

2.141.2.4 UeiDaqAPI void UeiDaq::CUeiSynchroResolverChannel::EnableExternalExcitation (bool *enabled*)

Enable or disable the external excitation.

Parameters:

enabled The new external excitation state

See also:

IsExternalExcitationEnabled (p. 502)

2.141.2.5 UeiDaqAPI double UeiDaq::CUeiSynchroResolverChannel::GetExcitationVoltage ()

Get the excitation RMS voltage configured for the channel.

Returns:

The excitation voltage.

See also:

SetExcitationVoltage (p. 503)

2.141.2.6 UeiDaqAPI void UeiDaq::CUeiSynchroResolverChannel::SetExcitationVoltage (double *vex*)

Set the excitation RMS voltage for this channel.

Parameters:

vex The excitation voltage

See also:

GetExcitationVoltage (p. 503)

2.141.2.7 UeiDaqAPI double UeiDaq::CUeiSynchroResolverChannel::GetExcitationFrequency ()

Get the excitation frequency configured for the channel.

Returns:

The excitation frequency.

See also:

SetExcitationFrequency (p. 503)

2.141.2.8 UeiDaqAPI void UeiDaq::CUeiSynchroResolverChannel::SetExcitationFrequency (double *fex*)

Set the excitation frequency for this channel.

Parameters:

fex The excitation frequency

See also:

GetExcitationFrequency (p. 503)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.142 UeiDaq::CUeiSynchroResolverWriter Class Reference

Synchro/Resolver Writer class.

`#include <UeiWriter.h>`

Public Member Functions

- UeiDaqAPI **CUeiSynchroResolverWriter** (**CUeiDataStream** *pDataStream)
  Constructor.
- virtual UeiDaqAPI ~**CUeiSynchroResolverWriter** ()
  Destructor.
- UeiDaqAPI void **WriteSingleAngle** (**f64** *pBuffer)
  Write angle of simulated synchro/resolver.
- UeiDaqAPI void **WriteMultipleAnglesToChannel** (int channel, int numberOfValues, **u-Int32** *pDelayBuffer, **f64** *pAngleBuffer)
  Write multiple angles to be generated with fixed delay to device FIFO.
- UeiDaqAPI void **Update** (**Int32** *written, **Int32** *available)
  Update all channels with staged angles and delays.

2.142.1 Detailed Description

Class that handles writing scaled data to a stream going to a simulated LVDT output

2.142.2 Constructor & Destructor Documentation

2.142.2.1 UeiDaqAPI UeiDaq::CUeiSynchroResolverWriter::CUeiSynchroResolverWriter (CUeiDataStream * *pDataStream*)

Parameters:

pDataStream represents the destination where to write data

2.142.3 Member Function Documentation

2.142.3.1 UeiDaqAPI void UeiDaq::CUeiSynchroResolverWriter::WriteSingleAngle (f64 * *pBuffer*)

Write one angle scan (one value per channel) to the output stream This is equivalent to CAnalog-ScaledWriter::WriteSingleScan()

Parameters:

pBuffer destination buffer

2.142.3.2 UeiDaqAPI void UeiDaq::CUeiSynchroResolverWriter::WriteMultipleAngles-ToChannel (int *channel*, int *numberOfValues*, uInt32 * *pDelayBuffer*, f64 * *pAngleBuffer*)

Write multiple angles to simulate velocity/acceleration on simulated synchro/resolver Each angle is associated with a delay (in microseconds) The angles are staged to be sent to the device. Actual send happens upon calling **Update()** (p. 505)

Parameters:

channel the channel to update

numberOfValues numberOfValues in the buffer

pDelayBuffer destination buffer for delays

pAngleBuffer destination buffer for angles

2.142.3.3 UeiDaqAPI void UeiDaq::CUeiSynchroResolverWriter::Update (Int32 * *written*, Int32 * *available*)

Update all channels with staged angles and delays

Parameters:

written array containing the number of values actually written for each channel

available array containing the space available in FIFO for each channel

The documentation for this class was generated from the following file:

- UeiWriter.h

2.143 UeiDaq::CUeiSystem Class Reference

This class contains static functions to retrieve informations about your system.

`#include <UeiSystem.h>`

Public Member Functions

- UeiDaqAPI **CUeiSystem** (void)
  Constructor.
- virtual UeiDaqAPI ~**CUeiSystem** (void)
  Destructor.

Static Public Member Functions

- static UeiDaqAPI **uInt32 GetFrameworkMajorVersion** ()
  Get the framework major version number.
- static UeiDaqAPI **uInt32 GetFrameworkMinorVersion** ()
  Get the framework minor version number.
- static UeiDaqAPI **uInt32 GetFrameworkExtraVersion** ()
  Get the framework extra version number.
- static UeiDaqAPI **uInt32 GetFrameworkBuildVersion** ()
  Get the framework build version number.
- static UeiDaqAPI std::string **GetFrameworkVersion** ()
  Get the framework version string.
- static UeiDaqAPI **uInt32 GetPluginLowLevelDriverMajorVersion** (std::string driverName)
  Get a driver plugin major version number.
- static UeiDaqAPI **uInt32 GetPluginLowLevelDriverMinorVersion** (std::string driverName)
  Get a driver plugin minor version number.
- static UeiDaqAPI std::string **GetPluginLowLevelDriverVersion** (std::string driverName)
  Get a driver plugin version string.
- static UeiDaqAPI std::string **GetFrameworkInstallationDirectory** ()
  Get the framework installation directory.
- static UeiDaqAPI std::string **GetPluginsInstallationDirectory** ()
  Get the framework plugins directory.

- static UeiDaqAPI std::string **GetSessionGroupDirectory** (std::string sessionGroup)
  Get a session group directory.
- static UeiDaqAPI **tUeiOSType GetOperatingSystemType** ()
  Get the operating system type.
- static UeiDaqAPI void **ReloadDrivers** ()
  Reload driver plugins.

2.143.1 Member Function Documentation

2.143.1.1 static UeiDaqAPI uInt32 UeiDaq::CUeiSystem::GetFrameworkMajorVersion () [static]

Get the first part of the four parts version number. major.minor.extra.build

Returns:

The major version number.

2.143.1.2 static UeiDaqAPI uInt32 UeiDaq::CUeiSystem::GetFrameworkMinorVersion () [static]

Get the second part of the four parts version number. major.minor.extra.build

Returns:

The minor version number.

2.143.1.3 static UeiDaqAPI uInt32 UeiDaq::CUeiSystem::GetFrameworkExtraVersion () [static]

Get the third part of the four parts version number. major.minor.extra.build

Returns:

The extra version number.

2.143.1.4 static UeiDaqAPI uInt32 UeiDaq::CUeiSystem::GetFrameworkBuildVersion () [static]

Get the fourth part of the four parts version number. major.minor.extra.build

Returns:

The build version number.

2.143.1.5 static UeiDaqAPI std::string UeiDaq::CUeiSystem::GetFrameworkVersion () [static]

Get the the four parts version string. "major.minor.extra.build"

Returns:

The version string.

2.143.1.6 static UeiDaqAPI uInt32 UeiDaq::CUeiSystem::GetPluginLowLevelDriverMajorVersion (std::string *driverName*) [static]

Get the first part of the four parts version number of the specified driver plugin. major.minor.extra.build

Parameters:

driverName The name of the driver plugin

Returns:

The major version number.

2.143.1.7 static UeiDaqAPI uInt32 UeiDaq::CUeiSystem::GetPluginLowLevelDriverMinorVersion (std::string *driverName*) [static]

Get the second part of the four parts version number of the specified driver plugin. major.minor.extra.build

Parameters:

driverName The name of the driver plugin

Returns:

The minor version number.

2.143.1.8 static UeiDaqAPI std::string UeiDaq::CUeiSystem::GetPluginLowLevelDriverVersion (std::string *driverName*) [static]

Get the the four parts version string of the specified driver plugin. "major.minor.extra.build"

Parameters:

driverName The name of the driver plugin

Returns:

The major version string.

2.143.1.9 static UeiDaqAPI std::string UeiDaq::CUeiSystem::GetFrameworkInstallationDirectory () [static]

Get the framework installation directory

Returns:

The framework installation directory.

2.143.1.10 static UeiDaqAPI std::string UeiDaq::CUeiSystem::GetPluginsInstallationDirectory () [static]

Get the framework plugins directory

Returns:

The framework plugins directory.

2.143.1.11 static UeiDaqAPI std::string UeiDaq::CUeiSystem::GetSessionGroupDirectory (std::string *sessionGroup*) [static]

Get a session group directory

Parameters:

sessionGroup the name of the session group

Returns:

The session group directory.

2.143.1.12 static UeiDaqAPI tUeiOSType UeiDaq::CUeiSystem::GetOperatingSystemType () [static]

Get the type of the operating system installed on this computer

Returns:

The operating system type.

2.143.1.13 static UeiDaqAPI void UeiDaq::CUeiSystem::ReloadDrivers () [static]

Un-load/re-load driver plugins

The documentation for this class was generated from the following file:

- UeiSystem.h

2.144 UeiDaq::CUeiTCChannel Class Reference

Manages settings for each thermocouple input channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiAIChannel**.

Public Member Functions

- UeiDaqAPI **CUeiTCChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiTCChannel** ()
  Destructor.
- UeiDaqAPI **tUeiThermocoupleType GetThermocoupleType** ()
  Get the thermocouple type.
- UeiDaqAPI void **SetThermocoupleType** (**tUeiThermocoupleType** tcType)
  Set the thermocouple type.
- UeiDaqAPI **tUeiTemperatureScale GetTemperatureScale** ()
  Get the temperature scale.
- UeiDaqAPI void **SetTemperatureScale** (**tUeiTemperatureScale** tempScale)
  Set the temperature scale.
- UeiDaqAPI **tUeiColdJunctionCompensationType GetCJCType** ()
  Get the cold junction compensation type.
- UeiDaqAPI void **SetCJCType** (**tUeiColdJunctionCompensationType** cjcType)
  Set the cold junction compensation type.
- UeiDaqAPI std::string **GetCJCResource** ()
  Get the cold junction compensation resource.
- UeiDaqAPI void **SetCJCResource** (std::string cjcResource)
  Set the cold junction compensation resource.
- UeiDaqAPI **UeiDaq::f64 GetCJCConstant** ()
  Get the cold junction compensation constant.
- UeiDaqAPI void **SetCJCConstant** (**UeiDaq::f64** cjcConstant)
  Set the cold junction compensation constant.

2.144.1 Detailed Description

Manages settings for each thermocouple input channel

2.144.2 Member Function Documentation

2.144.2.1 UeiDaqAPI tUeiThermocoupleType UeiDaq::CUeiTCChannel::GetThermocoupleType ()

Get the type of the thermocouple connected to the channel.

Returns:

the thermocouple type.

See also:

SetThermocoupleType (p. 511)

2.144.2.2 UeiDaqAPI void UeiDaq::CUeiTCChannel::SetThermocoupleType (tUeiThermocoupleType *tcType*)

Set the type of the thermocouple connected to the channel.

Parameters:

tcType the thermocouple type.

See also:

GetThermocoupleType (p. 511)

2.144.2.3 UeiDaqAPI tUeiTemperatureScale UeiDaq::CUeiTCChannel::GetTemperatureScale ()

Get the temperature scale used to convert the measured voltage to temperature.

Returns:

the temperature scale.

See also:

SetTemperatureScale (p. 511)

2.144.2.4 UeiDaqAPI void UeiDaq::CUeiTCChannel::SetTemperatureScale (tUeiTemperatureScale *tempScale*)

Set the temperature scale used to convert the measured voltage to temperature.

Parameters:

tempScale the temperature scale.

See also:

GetTemperatureScale (p. 511)

2.144.2.5 UeiDaqAPI tUeiColdJunctionCompensationType UeiDaq::CUeiTCChannel::GetCJCType ()

Get the type of the cold junction compensation used to convert from volts to temperature.

Returns:

the cold junction compensation type.

See also:

SetCJCType (p. 512)

2.144.2.6 UeiDaqAPI void UeiDaq::CUeiTCChannel::SetCJCType (tUeiColdJunctionCompensationType *cjcType*)

Set the type of the cold junction compensation used to convert from volts to temperature.

Parameters:

cjcType the cold junction compensation type.

See also:

GetCJCType (p. 512)

2.144.2.7 UeiDaqAPI std::string UeiDaq::CUeiTCChannel::GetCJCResource ()

Get the resource string describing how to measure the cold junction temperature. This setting is only used when the CJC type is set to UeiCJCTypeResource. The CJC resource format is similar to the channel list format: [cjcSensorType]://[Channel(s)]?[Paramters] cjcSensorType can be UTR, IC or AZ

UTR (Universal temperature reference), CJC is measured from an UTR RTD connected on the specified channel(s) Resource format is utr://0,1?A=65.3835,B=22.4752,C=498.063 where A,B and C are the RTD's coefficients for the Callendar Van-Dusen equation

IC: CJC is measured from an IC temperature sensor connected on the specified channel Resource format is ic://24?K=0.00295 where K is the sensor's coefficient

AZ: CJC is measured from an IC temperature sensor and Autozero offset is measured from a specified channel Resource format is az://23,24?K=0.00295 where K is the sensor's coefficient

Example: To measure CJC from a linear IC sensor connected to channel 20 the resource string is "ic://20?K=0.00295" this will measure the voltage on channel 20 and compute the CJC temperature with the formula: degK = V / 0.00295

Returns:

the cold junction compensation resource.

See also:

SetCJCResource (p. 513)

2.144.2.8 UeiDaqAPI void UeiDaq::CUeiTCChannel::SetCJCResource (std::string *cjcResource*)

Set the resource string describing how to measure the cold junction temperature. This setting is only used when the CJC type is set to UeiCJCTypeResource. The CJC resource format is similar to the channel list format: [cjcSensorType]://[Channel(s)]?[Paramters] cjcSensorType can be UTR, IC or AZ

UTR (Universal temperature reference), CJC is measured from an UTR RTD connected on the specified channel(s) Resource format is utr://0,1?A=65.3835,B=22.4752,C=498.063 where A,B and C are the RTD's coefficients for the Callendar Van-Dusen equation

IC: CJC is measured from an IC temperature sensor connected on the specified channel Resource format is ic://24?K=0.00295 where K is the sensor's coefficient

AZ: CJC is measured from an IC temperature sensor and Autozero offset is measured from a specified channel Resource format is az://23,24?K=0.00295 where K is the sensor's coefficient

Example: To measure CJC from a linear IC sensor connected to channel 20 the resource string is "ic://20?K=0.00295" this will measure the voltage on channel 20 and compute the CJC temperature with the formula: degK = V / 0.00295

Parameters:

cjcResource the cold junction compensation resource.

See also:

GetCJCResource (p. 512)

2.144.2.9 UeiDaqAPI UeiDaq::f64 UeiDaq::CUeiTCChannel::GetCJCConstant ()

Get the cold junction compensation temperature constant. This setting is only used when the CJC type is set to UeiCJCTypeConstant. The unit is the same as the configured temperature scale.

Returns:

the cold junction compensation constant.

See also:

SetCJCConstant (p. 513)

2.144.2.10 UeiDaqAPI void UeiDaq::CUeiTCChannel::SetCJCConstant (UeiDaq::f64 *cjcConstant*)

Set the cold junction compensation temperature constant. This setting is only used when the CJC type is set to UeiCJCTypeConstant. The unit is the same as the configured temperature scale.

Parameters:

cjcConstant the cold junction compensation constant.

See also:

GetCJCConstant (p. 513)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.145 UeiDaq::CUeiTimestampChannel Class Reference

Manages settings for each timestamp channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Public Member Functions

- UeiDaqAPI **CUeiTimestampChannel** ()

  Constructor.

- virtual UeiDaqAPI ~**CUeiTimestampChannel** ()

  Destructor.

- UeiDaqAPI **f64 GetResolution** ()

  Get the timestamp resolution.

- UeiDaqAPI void **SetResolution** (**f64** resolution)

  Set the timestamp resolution.

- UeiDaqAPI **f64 GetInitialTime** ()

  Get the initial time offset.

- UeiDaqAPI void **SetInitialTime** (**f64** initialTimeSeconds)

  Set the initial time offset.

2.145.1 Detailed Description

Manages settings for each timestamp channel

2.145.2 Member Function Documentation

2.145.2.1 UeiDaqAPI f64 UeiDaq::CUeiTimestampChannel::GetResolution ()

Get the timestamp resolution in seconds.

Returns:

the timestamp resolution.

See also:

SetResolution (p. 516)

2.145.2.2 UeiDaqAPI void UeiDaq::CUeiTimestampChannel::SetResolution (f64 *resolution*)

Set the timestamp resolution in seconds.

Parameters:

resolution the new resolution.

See also:

GetResolution (p. 515)

2.145.2.3 UeiDaqAPI f64 UeiDaq::CUeiTimestampChannel::GetInitialTime ()

A device timestamp channel start counting at 0 by default. The initial time offset is the decimal number of seconds since 00:00:00 UTC 01/01/1970 that will be added to the hardware timestamp.

Returns:

the initial time offset in seconds.

See also:

SetInitialTime (p. 516)

2.145.2.4 UeiDaqAPI void UeiDaq::CUeiTimestampChannel::SetInitialTime (f64 *initialTimeSeconds*)

A device timestamp channel start counting at 0 by default. The initial time offset is the decimal number of seconds since 00:00:00 UTC 01/01/1970 that will be added to the hardware timestamp.

Parameters:

initialTimeSeconds the new initial time offset in seconds.

See also:

GetInitialTime (p. 516)

The documentation for this class was generated from the following file:

- UeiChannel.h

2.146 UeiDaq::CUeiTiming Class Reference

Timing object.

`#include <UeiTiming.h>`

Inherits **UeiDaq::CUeiObject**.

Public Member Functions

- UeiDaqAPI **CUeiTiming** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiTiming** ()
  Destructor.
- UeiDaqAPI **tUeiTimingMode GetMode** ()
  Get the timing mode.
- UeiDaqAPI void **SetMode** (**tUeiTimingMode** mode)
  Set the timing mode.
- UeiDaqAPI **tUeiTimingClockSource GetConvertClockSource** ()
  Get the Conversion clock source.
- UeiDaqAPI void **SetConvertClockSource** (**tUeiTimingClockSource** source)
  Set the Conversion clock source.
- UeiDaqAPI **tUeiTimingClockSource GetScanClockSource** ()
  Get the Scan clock source.
- UeiDaqAPI void **SetScanClockSource** (**tUeiTimingClockSource** source)
  Set the Scan clock source.
- UeiDaqAPI **tUeiTimingClockSource GetTimestampClockSource** ()
  Get the timestamp clock source.
- UeiDaqAPI void **SetTimestampClockSource** (**tUeiTimingClockSource** source)
  Set the timestamp clock source.
- UeiDaqAPI **f64 GetConvertClockRate** ()
  Get the conversion clock rate.
- UeiDaqAPI void **SetConvertClockRate** (**f64** rate)
  Set the conversion clock rate.
- UeiDaqAPI **f64 GetScanClockRate** ()
  Get the scan clock rate.
- UeiDaqAPI void **SetScanClockRate** (**f64** rate)

Set the scan clock rate.

- UeiDaqAPI **tUeiDigitalEdge GetConvertClockEdge** ()
  Get the conversion clock edge.
- UeiDaqAPI void **SetConvertClockEdge** (**tUeiDigitalEdge** edge)
  Set the conversion clock edge.
- UeiDaqAPI **tUeiDigitalEdge GetScanClockEdge** ()
  Get the scan clock edge.
- UeiDaqAPI void **SetScanClockEdge** (**tUeiDigitalEdge** edge)
  Set the scan clock edge.
- UeiDaqAPI **tUeiTimingDuration GetDuration** ()
  Get the duration.
- UeiDaqAPI void **SetDuration** (**tUeiTimingDuration** duration)
  Set the duration.
- UeiDaqAPI **Int32 GetScanClockTimebaseDivisor** ()
  Get the scan clock divisor.
- UeiDaqAPI void **SetScanClockTimebaseDivisor** (**Int32** divisor)
  Set the scan clock divisor.
- UeiDaqAPI **Int32 GetConvertClockTimebaseDivisor** ()
  Get the convert clock divisor.
- UeiDaqAPI void **SetConvertClockTimebaseDivisor** (**Int32** divisor)
  Set the convert clock divisor.
- UeiDaqAPI **Int32 GetScanClockTimebaseDividingCounter** ()
  Get the scan clock dividing counter.
- UeiDaqAPI void **SetScanClockTimebaseDividingCounter** (**Int32** counter)
  Set the scan clock dividing counter.
- UeiDaqAPI **Int32 GetConvertClockTimebaseDividingCounter** ()
  Get the convert clock dividing counter.
- UeiDaqAPI void **SetConvertClockTimebaseDividingCounter** (**Int32** counter)
  Set the convert clock dividing counter.
- UeiDaqAPI std::string **GetScanClockSourceSignal** ()
  Get the name of the signal to use as Scan clock.
- UeiDaqAPI void **SetScanClockSourceSignal** (std::string signal)
  Set the name of the signal to use as Scan clock.

- UeiDaqAPI std::string **GetScanClockDestinationSignal** ()
  Get the name of the signal to route the Scan clock to.
- UeiDaqAPI void **SetScanClockDestinationSignal** (std::string signal)
  Set the name of the signal to route the Scan clock to.
- UeiDaqAPI std::string **GetConvertClockSourceSignal** ()
  Get the name of the signal to use as conversion clock.
- UeiDaqAPI void **SetConvertClockSourceSignal** (std::string signal)
  Set the name of the signal to use as conversion clock.
- UeiDaqAPI std::string **GetConvertClockDestinationSignal** ()
  Get the name of the signal to route the conversion clock to.
- UeiDaqAPI void **SetConvertClockDestinationSignal** (std::string signal)
  Set the name of the signal to route the conversion clock to.
- UeiDaqAPI **Int32 GetTimeout** ()
  Get the read or write timeout.
- UeiDaqAPI void **SetTimeout** (**Int32** timeout)
  Set the timeout.
- UeiDaqAPI std::string **GetTimestampSourceSignal** ()
  Get Timestamp source.
- UeiDaqAPI void **SetTimestampSourceSignal** (std::string signal)
  Set Timestamp source.
- UeiDaqAPI **f64 GetTimestampResolution** ()
  Get the timestamp resolution.
- UeiDaqAPI void **SetTimestampResolution** (**f64** resolution)
  Set the timestamp resolution.
- UeiDaqAPI bool **IsTimestampResetEnabled** ()
  Get the timestamp reset all state.
- UeiDaqAPI void **EnableTimestampReset** (bool enable)
  Enable/disable timestamp reset.
- UeiDaqAPI int **GetAsyncWatermark** ()
  Get the watermark level.
- UeiDaqAPI void **SetAsyncWatermark** (int watermark)
  Set the watermark level.
- UeiDaqAPI int **GetAsyncOutputWatermark** ()
  Get the output watermark level.

- UeiDaqAPI void **SetAsyncOutputWatermark** (int watermark)
  Set the output watermark level.
- UeiDaqAPI int **GetAsyncPeriod** ()
  Get the asynchronous period.
- UeiDaqAPI void **SetAsyncPeriod** (int periodUs)
  Set the asynchronous period.
- UeiDaqAPI int **GetAsyncNoActivityTimeout** ()
  Get the asynchronous no activity timeout.
- UeiDaqAPI void **SetAsyncNoActivityTimeout** (int timeoutUs)
  Set the asynchronous no activity timeout.
- UeiDaqAPI int **GetAsyncMaxDataSize** ()
  Get the max amount of messages returned along a timeout or periodic event.
- UeiDaqAPI void **SetAsyncMaxDatasize** (int maxDataSize)
  Set the max amount of messages returned along a timeout or periodic event.

2.146.1 Detailed Description

This object store the timing parameters of a session

2.146.2 Member Function Documentation

2.146.2.1 UeiDaqAPI tUeiTimingMode UeiDaq::CUeiTiming::GetMode ()

Get the timing mode

Returns:

the timing mode

See also:

SetMode (p. 520)

2.146.2.2 UeiDaqAPI void UeiDaq::CUeiTiming::SetMode (tUeiTimingMode *mode*)

Set the timing mode

Parameters:

mode the timing mode

See also:

GetMode (p. 520)

2.146.2.3 UeiDaqAPI tUeiTimingClockSource UeiDaq::CUeiTiming::GetConvertClockSource ()

Get the source of the A/D conversion clock

Returns:

the clock source

See also:

SetConvertClockSource (p. 521)

2.146.2.4 UeiDaqAPI void UeiDaq::CUeiTiming::SetConvertClockSource (tUeiTimingClockSource *source*)

Set the source of the A/D conversion clock

Parameters:

source the clock source

See also:

GetConvertClockSource (p. 521)

2.146.2.5 UeiDaqAPI tUeiTimingClockSource UeiDaq::CUeiTiming::GetScanClockSource ()

Get the source of the scan clock

Returns:

the clock source

See also:

SetScanClockSource (p. 521)

2.146.2.6 UeiDaqAPI void UeiDaq::CUeiTiming::SetScanClockSource (tUeiTimingClockSource *source*)

Set the source of the scan clock

Parameters:

source the clock source

See also:

GetScanClockSource (p. 521)

2.146.2.7 UeiDaqAPI tUeiTimingClockSource UeiDaq::CUeiTiming::GetTimestampClockSource ()

Get the source of the clock used to increment timestamp counter

Returns:

the timetamp clock source

See also:

SetTimestampClockSource (p. 522)

2.146.2.8 UeiDaqAPI void UeiDaq::CUeiTiming::SetTimestampClockSource (tUeiTimingClockSource *source*)

Set the source of the signal used to increment timestamp counter

Parameters:

source the timestamp clock source

See also:

GetTimestampClockSource (p. 522)

2.146.2.9 UeiDaqAPI f64 UeiDaq::CUeiTiming::GetConvertClockRate ()

Get the rate of the conversion clock

Returns:

the clock rate

See also:

SetConvertClockRate (p. 522)

2.146.2.10 UeiDaqAPI void UeiDaq::CUeiTiming::SetConvertClockRate (f64 *rate*)

Set the rate of the conversion clock

Parameters:

rate the clock rate

See also:

GetConvertClockRate (p. 522)

2.146.2.11 UeiDaqAPI f64 UeiDaq::CUeiTiming::GetScanClockRate ()

Get the rate of the scan clock

Returns:

the clock rate

See also:

SetScanClockRate (p. 523)

2.146.2.12 UeiDaqAPI void UeiDaq::CUeiTiming::SetScanClockRate (f64 *rate*)

Set the rate of the scan clock

Parameters:

rate the clock rate

See also:

GetScanClockRate (p. 523)

2.146.2.13 UeiDaqAPI tUeiDigitalEdge UeiDaq::CUeiTiming::GetConvertClockEdge ()

Get the active edge of the conversion clock.

Returns:

the clock active edge

See also:

SetConvertClockEdge (p. 523)

2.146.2.14 UeiDaqAPI void UeiDaq::CUeiTiming::SetConvertClockEdge (tUeiDigitalEdge *edge*)

Set the active edge of the conversion clock.

Parameters:

edge the clock active edge

See also:

GetConvertClockEdge (p. 523)

2.146.2.15 UeiDaqAPI tUeiDigitalEdge UeiDaq::CUeiTiming::GetScanClockEdge ()

Get the active edge of the scan clock.

Returns:

the clock active edge

See also:

SetScanClockEdge (p. 524)

2.146.2.16 UeiDaqAPI void UeiDaq::CUeiTiming::SetScanClockEdge (tUeiDigitalEdge *edge*)

Set the active edge of the scan clock.

Parameters:

edge the clock active edge

See also:

GetScanClockEdge (p. 524)

2.146.2.17 UeiDaqAPI tUeiTimingDuration UeiDaq::CUeiTiming::GetDuration ()

Get the duration of the timed operation.

Returns:

the duration

See also:

SetDuration (p. 524)

2.146.2.18 UeiDaqAPI void UeiDaq::CUeiTiming::SetDuration (tUeiTimingDuration *duration*)

Set the duration of the timed operation.

Parameters:

duration the duration

See also:

GetDuration (p. 524)

2.146.2.19 UeiDaqAPI Int32 UeiDaq::CUeiTiming::GetScanClockTimebaseDivisor ()

Get the scan clock divisor. This divisor is used to divide an external clock using one of the on-board counter/timers.

Returns:

the timebase divisor

See also:

SetScanClockTimebaseDivisor (p. 525)

2.146.2.20 UeiDaqAPI void UeiDaq::CUeiTiming::SetScanClockTimebaseDivisor (Int32 *divisor*)

Set the scan clock divisor. This divisor is used to divide an external clock using one of the on-board counter/timers.

Parameters:

divisor the timebase divisor

See also:

GetScanClockTimebaseDivisor (p. 525)

2.146.2.21 UeiDaqAPI Int32 UeiDaq::CUeiTiming::GetConvertClockTimebaseDivisor ()

Get the convert clock divisor. This divisor is used to divide an external clock using one of the on-board counter/timers.

Returns:

the timebase divisor

See also:

SetScanClockTimebaseDivisor (p. 525)

2.146.2.22 UeiDaqAPI void UeiDaq::CUeiTiming::SetConvertClockTimebaseDivisor (Int32 *divisor*)

Set the convert clock divisor. This divisor is used to divide an external clock using one of the on-board counter/timers.

Parameters:

divisor the timebase divisor

See also:

GetScanClockTimebaseDivisor (p. 525)

2.146.2.23 UeiDaqAPI Int32 UeiDaq::CUeiTiming::GetScanClockTimebaseDividingCounter ()

Get the scan clock dividing counter. This counter is used to divide an external clock signal.

Returns:

the timebase dividing counter

See also:

SetScanClockTimebaseDividingCounter (p. 526)

2.146.2.24 UeiDaqAPI void UeiDaq::CUeiTiming::SetScanClockTimebaseDividingCounter (Int32 *counter*)

Set the scan clock dividing counter. This counter is used to divide an external clock.

Parameters:

counter the timebase dividing counter

See also:

GetScanClockTimebaseDividingCounter (p. 526)

2.146.2.25 UeiDaqAPI Int32 UeiDaq::CUeiTiming::GetConvertClockTimebaseDividing-Counter ()

Get the convert clock dividing counter. This counter is used to divide an external clock.

Returns:

the timebase dividing counter

See also:

SetConvertClockTimebaseDividingCounter (p. 526)

2.146.2.26 UeiDaqAPI void UeiDaq::CUeiTiming::SetConvertClockTimebaseDividing-Counter (Int32 *counter*)

Set the convert clock dividing counter. This counter is used to divide an external.

Parameters:

counter the timebase dividing counter

See also:

GetConvertClockTimebaseDividingCounter (p. 526)

2.146.2.27 UeiDaqAPI std::string UeiDaq::CUeiTiming::GetScanClockSourceSignal ()

Get the name of the signal to use as Scan clock The signal name is device dependent.

Returns:

the current scan clock source signal

See also:

SetScanClockSourceSignal (p. 527)

2.146.2.28 UeiDaqAPI void UeiDaq::CUeiTiming::SetScanClockSourceSignal (std::string *signal*)

Set the name of the signal to use as Scan clock. The signal name is device dependent.

Parameters:

signal the new scan clock source signal

See also:

GetScanClockSourceSignal (p. 527)

2.146.2.29 UeiDaqAPI std::string UeiDaq::CUeiTiming::GetScanClockDestinationSignal ()

Get the name of the signal driven by the Scan clock The signal name is device dependent.

Returns:

the current scan clock destination signal

See also:

SetScanClockDestinationSignal (p. 527)

2.146.2.30 UeiDaqAPI void UeiDaq::CUeiTiming::SetScanClockDestinationSignal (std::string *signal*)

Set the name of the signal driven by the Scan clock. The signal name is device dependent.

Parameters:

signal the new scan clock destination signal

See also:

GetScanClockDestinationSignal (p. 527)

2.146.2.31 UeiDaqAPI std::string UeiDaq::CUeiTiming::GetConvertClockSourceSignal ()

Get the name of the signal to use as conversion clock The signal name is device dependent.

Returns:

the current conversion clock signal source

See also:

SetConvertClockSourceSignal (p. 528)

2.146.2.32 UeiDaqAPI void UeiDaq::CUeiTiming::SetConvertClockSourceSignal (std::string *signal*)

Set the name of the signal to use as conversion clock. The signal name is device dependent.

Parameters:

signal the new conversion clock signal source

See also:

GetConvertClockSourceSignal (p. 528)

2.146.2.33 UeiDaqAPI std::string UeiDaq::CUeiTiming::GetConvertClockDestinationSignal ()

Get the name of the signal driven by the conversion clock The signal name is device dependent.

Returns:

the current conversion clock signal destination

See also:

SetConvertClockDestinationSignal (p. 528)

2.146.2.34 UeiDaqAPI void UeiDaq::CUeiTiming::SetConvertClockDestinationSignal (std::string *signal*)

Set the name of the signal driven by the conversion clock The signal name is device dependent.

Parameters:

signal the new conversion clock signal destination

See also:

GetConvertClockDestinationSignal (p. 528)

2.146.2.35 UeiDaqAPI Int32 UeiDaq::CUeiTiming::GetTimeout ()

Get the maximum amount of time (in ms) for a read or write operation to complete.

Returns:

the timeout

See also:

SetTimeout (p. 529)

2.146.2.36 UeiDaqAPI void UeiDaq::CUeiTiming::SetTimeout (Int32 *timeout*)

Set the maximum amount of time (in ms) for a read or write operation to complete.

Parameters:

timeout the timeout

See also:

GetTimeout (p. 529)

Examples:

AnalogInOneShot_Trig_Ex.cpp, and **DigitalInEvent.cpp**.

2.146.2.37 UeiDaqAPI std::string UeiDaq::CUeiTiming::GetTimestampSourceSignal ()

Get the source of the signal used to increment timestamp counter

See also:

SetTimestampSourceSignal (p. 529)

2.146.2.38 UeiDaqAPI void UeiDaq::CUeiTiming::SetTimestampSourceSignal (std::string *signal*)

Set the source of the signal used to increment timestamp counter

See also:

GetTimestampSourceSignal (p. 529)

2.146.2.39 UeiDaqAPI f64 UeiDaq::CUeiTiming::GetTimestampResolution ()

Get the timestamp resolution in seconds.

Returns:

the timestamp resolution.

See also:

SetTimestampResolution (p. 530)

2.146.2.40 UeiDaqAPI void UeiDaq::CUeiTiming::SetTimestampResolution (f64 *resolution*)

Set the timestamp resolution in seconds.

Parameters:

resolution the new resolution.

See also:

GetTimestampResolution (p. 529)

2.146.2.41 UeiDaqAPI bool UeiDaq::CUeiTiming::IsTimestampResetEnabled ()

When enabled, all I/O layers timestamp counters will be reset at the same time when this session is started

Returns:

true if timestamp reset is enabled, false otherwise.

See also:

EnableTimestampReset (p. 530)

2.146.2.42 UeiDaqAPI void UeiDaq::CUeiTiming::EnableTimestampReset (bool *enable*)

When enabled, all I/O layers timestamp counters will be reset at the same time when this session is started

Parameters:

true to enable timestamp reset. false to disable.

See also:

IsTimestampResetEnabled (p. 530)

2.146.2.43 UeiDaqAPI int UeiDaq::CUeiTiming::GetAsyncWatermark ()

Get the watermark level used to configure FIFO asynchronous event.

Returns:

the watermark level.

2.146.2.44 UeiDaqAPI void UeiDaq::CUeiTiming::SetAsyncWatermark (int *watermark*)

Set the watermark level used to configure FIFO asynchronous event. (-1) to disable

Parameters:

watermark the new asynchronous watermark level.

2.146.2.45 UeiDaqAPI int UeiDaq::CUeiTiming::GetAsyncOutputWatermark ()

Get the output watermark level used to configure FIFO asynchronous event.

Returns:

the output watermark level.

2.146.2.46 UeiDaqAPI void UeiDaq::CUeiTiming::SetAsyncOutputWatermark (int *watermark*)

Set the output watermark level used to configure FIFO asynchronous event. (-1) to disable

Parameters:

watermark the new asynchronous watermark level.

2.146.2.47 UeiDaqAPI int UeiDaq::CUeiTiming::GetAsyncPeriod ()

Get the asynchronous period used to fire events when FIFO level is below watermark.

Returns:

the period in microseconds.

2.146.2.48 UeiDaqAPI void UeiDaq::CUeiTiming::SetAsyncPeriod (int *periodUs*)

Set the asynchronous period used to fire events when FIFO level is below watermark. (-1 to disable)

Parameters:

periodUs the new period in microseconds.

2.146.2.49 UeiDaqAPI int UeiDaq::CUeiTiming::GetAsyncNoActivityTimeout ()

Get the asynchronous timeout to fire events when no activity has been detected for the specified amount of time.

Returns:

the period in microseconds.

2.146.2.50 UeiDaqAPI void UeiDaq::CUeiTiming::SetAsyncNoActivityTimeout (int *timeoutUs*)

Set the asynchronous timeout to fire events when no activity has been detected for the specified amount of time. (-1 to disable)

Parameters:

timeoutUs the new timeout in microseconds.

2.146.2.51 UeiDaqAPI int UeiDaq::CUeiTiming::GetAsyncMaxDataSize ()

Get the max amount of messages returned along a timeout of periodic event.

Returns:

the max amount of data .

2.146.2.52 UeiDaqAPI void UeiDaq::CUeiTiming::SetAsyncMaxDatasize (int *maxDataSize*)

Set the max amount of messages returned along a timeout of periodic event.

Parameters:

maxDataSize the new timeout in microseconds.

The documentation for this class was generated from the following file:

- UeiTiming.h

2.147 UeiDaq::CUeiTrigger Class Reference

Trigger object.

`#include <UeiTrigger.h>`

Inherits **UeiDaq::CUeiObject**.

Public Member Functions

- UeiDaqAPI **CUeiTrigger** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiTrigger** ()
  Destructor.
- UeiDaqAPI **tUeiTriggerSource GetTriggerSource** ()
  Get the trigger source.
- UeiDaqAPI void **SetTriggerSource** (**tUeiTriggerSource** src)
  Set the trigger source.
- UeiDaqAPI **tUeiDigitalEdge GetDigitalEdge** ()
  Get the trigger digital edge.
- UeiDaqAPI void **SetDigitalEdge** (**tUeiDigitalEdge** edge)
  Set the trigger digital edge.
- UeiDaqAPI std::string **GetSourceSignal** ()
  Get the trigger source signal.
- UeiDaqAPI void **SetSourceSignal** (std::string signal)
  Set the trigger source signal.
- UeiDaqAPI std::string **GetDestinationSignal** ()
  Get the trigger destination signal.
- UeiDaqAPI void **SetDestinationSignal** (std::string signal)
  Set the trigger destination signal.
- UeiDaqAPI **tUeiTriggerAction GetAction** ()
  Get the trigger action.
- UeiDaqAPI void **SetAction** (**tUeiTriggerAction** action)
  Set the trigger action.
- UeiDaqAPI **tUeiTriggerCondition GetCondition** ()
  Get the analog trigger condition.
- UeiDaqAPI void **SetCondition** (**tUeiTriggerCondition** condition)

Set the analog trigger condition.

- UeiDaqAPI **Int32 GetChannel** ()
  Get the trigger channel.
- UeiDaqAPI void **SetChannel** (**Int32** channel)
  Set the trigger channel.
- UeiDaqAPI **f64 GetLevel** ()
  Get the trigger level.
- UeiDaqAPI void **SetLevel** (**f64** level)
  Set the trigger level.
- UeiDaqAPI **f64 GetHysteresis** ()
  Get the trigger hysteresis.
- UeiDaqAPI void **SetHysteresis** (**f64** hysteresis)
  Set the trigger hysteresis.
- UeiDaqAPI **Int32 GetNumberOfPreTriggerScans** ()
  Get the number of pre-trigger scans.
- UeiDaqAPI void **SetNumberOfPreTriggerScans** (**Int32** numPreTriggerScans)
  Set the number of pre-trigger scans.
- UeiDaqAPI void **Fire** ()
  Manually send a trigger.

2.147.1 Detailed Description

This object store the trigger parameters of a session

2.147.2 Member Function Documentation

2.147.2.1 UeiDaqAPI tUeiTriggerSource UeiDaq::CUeiTrigger::GetTriggerSource ()

Get the source of the signal that will trigger the start of the session

Returns:

the current trigger source

See also:

SetTriggerSource (p. 535)

2.147.2.2 UeiDaqAPI void UeiDaq::CUeiTrigger::SetTriggerSource (tUeiTriggerSource *src*)

Set the source of the signal that will trigger the start of the session

Parameters:

src the new trigger source

See also:

GetTriggerSource (p. 534)

2.147.2.3 UeiDaqAPI tUeiDigitalEdge UeiDaq::CUeiTrigger::GetDigitalEdge ()

Get the active edge of the trigger signal

Returns:

the current trigger edge

See also:

SetDigitalEdge (p. 535)

2.147.2.4 UeiDaqAPI void UeiDaq::CUeiTrigger::SetDigitalEdge (tUeiDigitalEdge *edge*)

Set the active edge of the trigger signal

Parameters:

edge the new trigger edge

See also:

GetDigitalEdge (p. 535)

2.147.2.5 UeiDaqAPI std::string UeiDaq::CUeiTrigger::GetSourceSignal ()

Get the signal used to transmit the trigger to the device. The signal name is device dependent.

Returns:

the current trigger source signal

See also:

SetSourceSignal (p. 536)

2.147.2.6 UeiDaqAPI void UeiDaq::CUeiTrigger::SetSourceSignal (std::string *signal*)

Set the signal used to transmit the trigger to the device. The signal name is device dependent.

Parameters:

signal the new trigger source signal

See also:

GetSourceSignal (p. 535)

2.147.2.7 UeiDaqAPI std::string UeiDaq::CUeiTrigger::GetDestinationSignal ()

Get the signal where the trigger is routed out of the device. The signal name is device dependent.

Returns:

the current trigger destination signal

See also:

SetDestinationSignal (p. 536)

2.147.2.8 UeiDaqAPI void UeiDaq::CUeiTrigger::SetDestinationSignal (std::string *signal*)

Set the signal where the trigger is routed out of the device. The signal name is device dependent.

Parameters:

signal the new trigger destination signal

See also:

GetSignal

2.147.2.9 UeiDaqAPI tUeiTriggerAction UeiDaq::CUeiTrigger::GetAction ()

Get the action to execute when the trigger will occur

Returns:

the current trigger action

See also:

SetAction (p. 537)

2.147.2.10 UeiDaqAPI void UeiDaq::CUeiTrigger::SetAction (tUeiTriggerAction *action*)

Set the action to execute when the trigger will occur

Parameters:

action the new trigger action

See also:

GetAction (p. 536)

2.147.2.11 UeiDaqAPI tUeiTriggerCondition UeiDaq::CUeiTrigger::GetCondition ()

Get the condition for the analog software trigger

Returns:

the current condition

See also:

SetCondition (p. 537)

2.147.2.12 UeiDaqAPI void UeiDaq::CUeiTrigger::SetCondition (tUeiTriggerCondition *condition*)

Set the condition for the analog software trigger

Parameters:

condition the new condition

See also:

GetCondition (p. 537)

2.147.2.13 UeiDaqAPI Int32 UeiDaq::CUeiTrigger::GetChannel ()

Get the channel to monitor for the trigger condition.

Returns:

the current channel to monitor.

See also:

SetChannel (p. 538)

2.147.2.14 UeiDaqAPI void UeiDaq::CUeiTrigger::SetChannel (Int32 *channel*)

Set the channel to monitor for the trigger condition.

Parameters:

channel the new channel to monitor

See also:

GetChannel (p. 537)

2.147.2.15 UeiDaqAPI f64 UeiDaq::CUeiTrigger::GetLevel ()

Get the scaled value at which the trigger occurs. The value is in the same unit as the measurement.

Returns:

the current level

See also:

SetLevel (p. 538)

2.147.2.16 UeiDaqAPI void UeiDaq::CUeiTrigger::SetLevel (f64 *level*)

Set the scaled value at which the trigger occurs. The value must be specified in the same unit as the measurement.

Parameters:

level the new level

See also:

GetLevel (p. 538)

2.147.2.17 UeiDaqAPI f64 UeiDaq::CUeiTrigger::GetHysteresis ()

Get the hysteresis window scaled value. The hysteresis window is a noise filter. The trigger occurs when the signal leave the window.

Returns:

the current hysteresis

See also:

SetHysteresis (p. 539)

2.147.2.18 UeiDaqAPI void UeiDaq::CUeiTrigger::SetHysteresis (f64 *hysteresis*)

Set the hysteresis window scaled value. The hysteresis window is a noise filter. The trigger occurs when the signal leave the window.

Parameters:

hysteresis the new hysteresis

See also:

GetHysteresis (p. 538)

2.147.2.19 UeiDaqAPI Int32 UeiDaq::CUeiTrigger::GetNumberOfPreTriggerScans ()

The number of scans saved before the trigger occurred.

Returns:

the number of pre-trigger scans

See also:

SetNumberOfPreTriggerScans (p. 539)

2.147.2.20 UeiDaqAPI void UeiDaq::CUeiTrigger::SetNumberOfPreTriggerScans (Int32 *numPreTriggerScans*)

The number of scans to save before the trigger occurs

Parameters:

numPreTriggerScans the new nunber of pre-trigger scans

See also:

GetNumberOfPreTriggerScans (p. 539)

2.147.2.21 UeiDaqAPI void UeiDaq::CUeiTrigger::Fire ()

Manually send a trigger event. This method can only be called When trigger source is set to UeiTriggerSourceSignal and the trigger signal is set to a software assertable signal.

The documentation for this class was generated from the following file:

- UeiTrigger.h

2.148 UeiDaq::CUeiVRChannel Class Reference

Manages settings for each Variable Reluctance channel.

`#include <UeiChannel.h>`

Inherits **UeiDaq::CUeiChannel**.

Public Member Functions

- UeiDaqAPI **CUeiVRChannel** ()
  Constructor.
- virtual UeiDaqAPI ~**CUeiVRChannel** ()
  Destructor.
- UeiDaqAPI **tUeiVRMode GetMode** ()
  Get the current VR channel input mode.
- UeiDaqAPI void **SetMode** (**tUeiVRMode** mode)
  Set the VR channel input mode.
- UeiDaqAPI **tUeiVRZCMode GetZCMode** ()
  Get the current zero-crossing mode.
- UeiDaqAPI void **SetZCMode** (**tUeiVRZCMode** mode)
  Set the zero-crossing mode.
- UeiDaqAPI **tUeiVRAPTMode GetAPTMode** ()
  Get the current adaptive peak threshold mode.
- UeiDaqAPI void **SetAPTMode** (**tUeiVRAPTMode** mode)
  Set the adaptive peak threshold mode.
- UeiDaqAPI double **GetADCRate** (void)
  Get the ADC measurement rate.
- UeiDaqAPI void **SetADCRate** (double rate)
  Set the ADC rate.
- UeiDaqAPI int **GetADCMovingAverage** (void)
  Get the ADC moving average.
- UeiDaqAPI void **SetADCMovingAverage** (int mvAvrg)
  Set the ADC moving average.
- UeiDaqAPI int **GetAPTThresholdDivider** (void)
  Get the APT threshold divider.
- UeiDaqAPI void **SetAPTThresholdDivider** (int divider)

Set the APT threshold divider.

- UeiDaqAPI double **GetAPTThreshold** (void)
  Get the APT threshold.
- UeiDaqAPI void **SetAPTThreshold** (double threshold)
  Set the APT threshold.
- UeiDaqAPI double **GetZCLevel** (void)
  Get the ZC level.
- UeiDaqAPI void **SetZCLevel** (double level)
  Set the ZC level.
- UeiDaqAPI int **GetNumberOfTeeth** (void)
  Get the number of teeth on the encoder wheel.
- UeiDaqAPI void **SetNumberOfTeeth** (int numTeeth)
  Set the number of teeth on the encoder wheel.
- UeiDaqAPI int **GetZToothSize** (void)
  Get the Z tooth size.
- UeiDaqAPI void **SetZToothSize** (int size)
  Set the Z tooth size.
- UeiDaqAPI **tUeiVRFIFOMode GetFIFOMode** (void)
  Get the optional FIFO mode.
- UeiDaqAPI void **SetFIFOMode** (**tUeiVRFIFOMode** fifoMode)
  Set the optional FIFO mode.
- UeiDaqAPI double **GetTimedModeRate** (void)
  Get the timed mode measurement rate.
- UeiDaqAPI void **SetTimedModeRate** (double rate)
  Set the timed mode measurement rate.

2.148.1 Detailed Description

Manages settings for VR channels uses to read data from a VR sensor. A VR input device converts the VR sensor analog signal to a digital signal that can be processed by a counter/timer

2.148.2 Member Function Documentation

2.148.2.1 UeiDaqAPI tUeiVRMode UeiDaq::CUeiVRChannel::GetMode ()

The VR-608 can use four different modes to measure velocity, position or direction The counting mode can be set to: Decoder: Even and Odd channels are used in pair to determine direction and

position Timed: Count number of teeth detected during a timed interval N pulses: Measure the time taken to detect N teeth (Number of teeth needs to be set separately) Z pulse: Measure the number of teeth and the time elapsed between two Z pulses (The Z tooth is usually a gap or a double tooth on the encoder wheel)

In decoder mode the settings for the even channel are applied to both even and odd channels.

Returns:

the current VR mode

2.148.2.2 UeiDaqAPI void UeiDaq::CUeiVRChannel::SetMode (tUeiVRMode *mode*)

The VR-608 can use four different modes to measure velocity, position or direction The counting mode can be set to: Decoder: Even and Odd channels are used in pair to determine direction and position Timed: Count number of teeth detected during a timed interval N pulses: Measure the time taken to detect N teeth (Number of teeth needs to be set separately) Z pulse: Measure the number of teeth and the time elapsed between two Z pulses (The Z tooth is usually a gap or a double tooth on the encoder wheel)

In decoder mode the settings for the even channel are applied to both even and odd channels.

Parameters:

mode the new VR mode

2.148.2.3 UeiDaqAPI tUeiVRZCMode UeiDaq::CUeiVRChannel::GetZCMode ()

Zero crossing finds the point in time where the VR sensor output voltage goes from positive to negative voltage. This point is when the center of the tooth is lining up with the center of the VR sensor.

Returns:

the current Zero Crossing mode

2.148.2.4 UeiDaqAPI void UeiDaq::CUeiVRChannel::SetZCMode (tUeiVRZCMode *mode*)

Zero crossing finds the point in time where the VR sensor output voltage goes from positive to negative voltage. This point is when the center of the tooth is lining up with the center of the VR sensor.

Parameters:

mode the new zero-crossing mode

2.148.2.5 UeiDaqAPI tUeiVRAPTMode UeiDaq::CUeiVRChannel::GetAPTMode ()

APT finds the point in time where the VR sensor output voltage falls below a certain threshold. This point marks the beginning of the gap between two teeth.

Returns:

the current APT mode

2.148.2.6 UeiDaqAPI void UeiDaq::CUeiVRChannel::SetAPTMode (tUeiVRAPTMode *mode*)

APT finds the point in time where the VR sensor output voltage falls below a certain threshold. This point marks the beginning of the gap between two teeth.

Parameters:

mode the new APT mode

2.148.2.7 UeiDaqAPI double UeiDaq::CUeiVRChannel::GetADCRate (void)

Get the rate at which the VR sensor signal is measured.

Returns:

The current ADC rate.

2.148.2.8 UeiDaqAPI void UeiDaq::CUeiVRChannel::SetADCRate (double *rate*)

Set the rate at which the VR sensor is measured.

Parameters:

rate The new ADC rate.

2.148.2.9 UeiDaqAPI int UeiDaq::CUeiVRChannel::GetADCMovingAverage (void)

Get the size of the moving average window applied to the VR sensor signal while it is measured.

Returns:

The current ADC moving average.

2.148.2.10 UeiDaqAPI void UeiDaq::CUeiVRChannel::SetADCMovingAverage (int *mvAvrg*)

Set the size of the moving average window applied to the VR sensor signal while it is measured.

Parameters:

mvAvrg The new ADC moving average.

2.148.2.11 UeiDaqAPI int UeiDaq::CUeiVRChannel::GetAPTThresholdDivider (void)

The APT threshold divider is used when APT mode is set to "Logic". It specifies that the AP threshold will be set at a fraction of the peak input voltage This is a value between 1 and 15: 1=1/2, 2=1/4, 3=1/8 etc...

Returns:

The current APT threshold divider.

2.148.2.12 UeiDaqAPI void UeiDaq::CUeiVRChannel::SetAPTThresholdDivider (int *divider*)

The APT threshold divider is used when APT mode is set to "Logic" It specifies that the AP threshold will be set at a fraction of the peak input voltage This is a value between 1 and 15: 1=1/2, 2=1/4, 3=1/8 etc...

Parameters:

divider The new APT threshold divider.

2.148.2.13 UeiDaqAPI double UeiDaq::CUeiVRChannel::GetAPTThreshold (void)

The APT threshold is used when APT mode is set to "Fixed"

Returns:

The current APT threshold.

2.148.2.14 UeiDaqAPI void UeiDaq::CUeiVRChannel::SetAPTThreshold (double *threshold*)

The APT threshold is used when APT mode is set to "Fixed"

Parameters:

threshold The new APT threshold.

2.148.2.15 UeiDaqAPI double UeiDaq::CUeiVRChannel::GetZCLevel (void)

The ZC level is used when ZC mode is set to "Fixed"

Returns:

The current ZC level.

2.148.2.16 UeiDaqAPI void UeiDaq::CUeiVRChannel::SetZCLevel (double *level*)

The ZC level is used when ZC mode is set to "Fixed"

Parameters:

level The new ZC level.

2.148.2.17 UeiDaqAPI int UeiDaq::CUeiVRChannel::GetNumberOfTeeth (void)

The number of teeth on the encoder wheel

Returns:

The current number of teeth.

2.148.2.18 UeiDaqAPI void UeiDaq::CUeiVRChannel::SetNumberOfTeeth (int *numTeeth*)

The number of teeth on the encoder wheel

Parameters:

numTeeth The new number of teeth.

2.148.2.19 UeiDaqAPI int UeiDaq::CUeiVRChannel::GetZToothSize (void)

A Z Tooth is usually materialized by a one or more missing teeth or one or more fused teeth. This parameter specified the number of fused or missing teeth.

Returns:

The current number Z tooth size.

2.148.2.20 UeiDaqAPI void UeiDaq::CUeiVRChannel::SetZToothSize (int *size*)

A Z Tooth is usually materialized by a one or more missing teeth or one or more fused teeth. This parameter specified the number of fused or missing teeth.

Parameters:

size The new Z tooth size.

2.148.2.21 UeiDaqAPI tUeiVRFIFOMode UeiDaq::CUeiVRChannel::GetFIFOMode (void)

When enabled, FIFO data can be used to calculate a map of the inter-tooth delays around the encoder wheel. this is useful to calculate acceleration

Returns:

The FIFO data status.

2.148.2.22 UeiDaqAPI void UeiDaq::CUeiVRChannel::SetFIFOMode (tUeiVRFIFOMode *fifoMode*)

When enabled, FIFO data can be used to calculate a map of the inter-tooth delays around the encoder wheel. this is useful to calculate acceleration

Parameters:

fifoMode The new FIFO mode.

2.148.2.23 UeiDaqAPI double UeiDaq::CUeiVRChannel::GetTimedModeRate (void)

Get the rate at which the VR sensor signal is measured.

Returns:

The current timed mode rate.

2.148.2.24 UeiDaqAPI void UeiDaq::CUeiVRChannel::SetTimedModeRate (double *rate*)

Set the rate at which the VR sensor is measured.

Parameters:

rate The new timed mode rate.

The documentation for this class was generated from the following file:

- UeiChannel.h

2.149 UeiDaq::CUeiVRReader Class Reference

Variable Reluctance reader class.

`#include <UeiMessaging.h>`

Public Member Functions

- UeiDaqAPI **CUeiVRReader** (**CUeiDataStream** *pDataStream, **Int32** channel=0)
  Constructor.
- virtual UeiDaqAPI ~**CUeiVRReader** ()
  Destructor.
- UeiDaqAPI void **Read** (**Int32** numVals, **tUeiVRData** *pBuffer, **Int32** *numValsRead)
  Read data from VR device.
- UeiDaqAPI void **Read** (**Int32** numValues, **uInt32** *pBuffer, **Int32** *numValuesRead)
  Read variable reluctance FIFO data.
- UeiDaqAPI void **Read** (**Int32** numVals, double *pBuffer, **Int32** *numValsRead)
  Read ADC data.
- UeiDaqAPI void **ReadADCStatus** (**uInt32** *pStatus)
  Read ADC status.

2.149.1 Detailed Description

Class that handles reading variable reluctance measurements from a stream associated with a VR input channel

2.149.2 Constructor & Destructor Documentation

2.149.2.1 UeiDaqAPI UeiDaq::CUeiVRReader::CUeiVRReader (CUeiDataStream * *pDataStream*, Int32 *channel* = 0)

Parameters:

pDataStream represents the source where to read data

channel the VR channel to read from

2.149.3 Member Function Documentation

2.149.3.1 UeiDaqAPI void UeiDaq::CUeiVRReader::Read (Int32 *numVals*, tUeiVRData * *pBuffer*, Int32 * *numValsRead*)

Read position, velocity, number of counted teeth and timestamp.

Parameters:

numVals the number of values to read

pBuffer destination buffer

numValsRead the actual number of values read

2.149.3.2 UeiDaqAPI void UeiDaq::CUeiVRReader::Read (Int32 *numValues*, uInt32 * *pBuffer*, Int32 * *numValuesRead*)

Variable reluctance FIFO data can be used to calculate a map of the inter-tooth delays around the encoder wheel. this is useful to calculate acceleration.

Parameters:

numValues number of values to read

pBuffer destination buffer

numValuesRead Number of values actually read

2.149.3.3 UeiDaqAPI void UeiDaq::CUeiVRReader::Read (Int32 *numVals*, double * *pBuffer*, Int32 * *numValsRead*)

Read ADC FIFO data, this is used as a diagnostic to visualize the shape of the signal as it is acquired on the VR input device. There is one FIFO for each pair of channel. this function reads the ADC data for both even and odd channels

Parameters:

numVals the number of values to read

pBuffer destination buffer for even and odd channel ADC data

numValsRead the actual number of values read

2.149.3.4 UeiDaqAPI void UeiDaq::CUeiVRReader::ReadADCStatus (uInt32 * *pStatus*)

Read ADC status, this is used as a diagnostic to troubleshoot VR inputs

Parameters:

pStatus destination buffer for even and odd channel ADC data

The documentation for this class was generated from the following file:

- UeiMessaging.h

2.150 UeiDaq::IUeiEventListener Class Reference

Interface called when an event occurs.

`#include <UeiEvent.h>`

Public Member Functions

- virtual void **OnEvent** (**tUeiEvent** event, void *param)=0
  Event callback method.

2.150.1 Detailed Description

Implement this interface to receive asynchronous notifications

Examples:

AnalogInBufferedAsync.cpp, and **AnalogOutBufferedAsync.cpp**.

2.150.2 Member Function Documentation

2.150.2.1 virtual void UeiDaq::IUeiEventListener::OnEvent (tUeiEvent *event*, void * *param*) [pure virtual]

This method is called by the framework when an event occurs

Parameters:

event Event that triggered the callback

param data pointer, dependent on the event

The documentation for this class was generated from the following file:

- UeiEvent.h

Chapter 3

UeiDaq Framework File Documentation

3.1 UeiConstants.h File Reference

Typedefs

- typedef long long **Int64**
  32-bit signed integer
- typedef unsigned long long **uInt64**
  32-bit unsigned integer
- typedef int **Int32**
  32-bit signed integer
- typedef unsigned int **uInt32**
  32-bit unsigned integer
- typedef short **Int16**
  16-bit signed integer
- typedef unsigned short **uInt16**
  16-bit unsigned integer
- typedef char **Int8**
  8-bit signed integer
- typedef unsigned char **uInt8**
  8-bit unsigned integer
- typedef float **f32**
  32-bit floating point
- typedef double **f64**

64-bit double precision floating point

- typedef enum **_tUeiAIChannelInputMode tUeiAIChannelInputMode**
  AI input mode.
- typedef enum **_tUeiAIChannelMuxPos tUeiAIChannelMuxPos**
  AI mux modes.
- typedef enum **_tUeiAOWaveformType tUeiAOWaveformType**
  AO waveform type.
- typedef enum **_tUeiAOWaveformMode tUeiAOWaveformMode**
  AO waveform mode.
- typedef enum **_tUeiAOWaveformXform tUeiAOWaveformXform**
  AO waveform transform.
- typedef enum **_tUeiAOWaveformOffsetClockSource tUeiAOWaveformOffsetClockSource**
  AO waveform offset DAC clock source.
- typedef enum **_tUeiAOWaveformClockSource tUeiAOWaveformClockSource**
  AO waveform main clock source.
- typedef enum **_tUeiAOWaveformClockSync tUeiAOWaveformClockSync**
  AO waveform clock synchronization.
- typedef enum **_tUeiAOWaveformOffsetTriggerSource tUeiAOWaveformOffsetTriggerSource**
  AO waveform offset DAC trigger source.
- typedef enum **_tUeiAOWaveformTriggerSource tUeiAOWaveformTriggerSource**
  AO waveform main DAC trigger source.
- typedef enum **_tUeiAOWaveformCommandType tUeiAOWaveformCommandType**
  AO waveform message type.
- typedef enum **_tUeiAOWaveformSweepControl tUeiAOWaveformSweepControl**
  AO waveform sweep control.
- typedef enum **_tUeiAODACMode tUeiAODACMode**
  DAC mode.
- typedef enum **_tUeiAODiagChannel tUeiAODiagChannel**
  AO Diagnostic ADC channel.
- typedef enum **_tUeiCounterMode tUeiCounterMode**
  Counter/Timer mode.
- typedef enum **_tUeiCounterSource tUeiCounterSource**

Counter/Timer source.

- typedef enum **_tUeiCounterGate tUeiCounterGate**
  Counter/Timer gate.
- typedef enum **_tUeiCounterGateMode tUeiCounterGateMode**
  Counter/Timer gate mode.
- typedef enum **_tUeiCounterCaptureTimebase tUeiCounterCaptureTimebase**
  Counter/Timer measurement timebase.
- typedef enum **_tUeiDigitalEdge tUeiDigitalEdge**
  Digital Edge.
- typedef enum **_tUeiDigitalInputMux tUeiDigitalInputMux**
  Digital Input Mux.
- typedef enum **_tUeiDigitalTermination tUeiDigitalTermination**
  Digital output termination.
- typedef enum **_tUeiTimingDuration tUeiTimingDuration**
  Timing Duration.
- typedef enum **_tUeiTimingMode tUeiTimingMode**
  Timing mode.
- typedef enum **_tUeiTimingClockSource tUeiTimingClockSource**
  Clock source.
- typedef enum **_tUeiSessionType tUeiSessionType**
  Session type.
- typedef enum **_tUeiSessionState tUeiSessionState**
  Session state.
- typedef enum **_tUeiEvent tUeiEvent**
  Asynchronous event.
- typedef enum **_tUeiTriggerSource tUeiTriggerSource**
  Trigger Source.
- typedef enum **_tUeiTriggerCondition tUeiTriggerCondition**
  Trigger Condition.
- typedef enum **_tUeiTriggerAction tUeiTriggerAction**
  Trigger Action.
- typedef enum **_tUeiDataStreamRelativeTo tUeiDataStreamRelativeTo**
  Buffer access reference.

- typedef enum **_tUeiThermocoupleType tUeiThermocoupleType**
  Thermocouple types.
- typedef enum **_tUeiColdJunctionCompensationType tUeiColdJunctionCompensationType**
  Cold junction sensor type.
- typedef enum **_tUeiTemperatureScale tUeiTemperatureScale**
  Temperature scales.
- typedef enum **_tUeiRTDType tUeiRTDType**
  RTD sensor type.
- typedef enum **_tUeiReferenceResistorType tUeiReferenceResistorType**
  Reference resistor type.
- typedef enum **_tUeiStrainGageBridgeType tUeiStrainGageBridgeType**
  Strain gage bridge configurations.
- typedef enum **_tUeiSensorBridgeType tUeiSensorBridgeType**
  Sensor bridge configurations.
- typedef enum **_tUeiMeasurementType tUeiMeasurementType**
  Measurement type.
- typedef enum **_tUeiOSType tUeiOSType**
  OS Type.
- typedef enum **_tUeiSerialPortMode tUeiSerialPortMode**
  Serial port mode.
- typedef enum **_tUeiSerialPortSpeed tUeiSerialPortSpeed**
  Serial port speed.
- typedef enum **_tUeiSerialPortDataBits tUeiSerialPortDataBits**
  Serial port number of data bits per character frame.
- typedef enum **_tUeiSerialPortParity tUeiSerialPortParity**
  Serial port parity.
- typedef enum **_tUeiSerialPortStopBits tUeiSerialPortStopBits**
  Serial port number of stop bits.
- typedef enum **_tUeiSerialPortFlowControl tUeiSerialPortFlowControl**
  Serial port flow control setting.
- typedef enum **_tUeiSerialPortMinorFrameMode tUeiSerialPortMinorFrameMode**
  Minor frame delay mode.
- typedef enum **_tUeiSerialReadDataType tUeiSerialReadDataType**

CUeiSerialReader data types.

- typedef enum **_tUeiCANPortSpeed tUeiCANPortSpeed**
  CAN port speed.
- typedef enum **_tUeiCANFrameFormat tUeiCANFrameFormat**
  CAN frame format.
- typedef enum **_tUeiCANFrameType tUeiCANFrameType**
  CAN frame type.
- typedef enum **_tUeiCANPortMode tUeiCANPortMode**
  CAN operation mode.
- typedef enum **_tUeiWheatstoneBridgeBranch tUeiWheatstoneBridgeBranch**
  Wheatstone bridge branches.
- typedef enum **_tUeiSignalSource tUeiSignalSource**
  The source of a synchronization signal.
- typedef enum **_tUeiWiringScheme tUeiWiringScheme**
  Sensor with excitation wiring schemes.
- typedef enum **_tUeiARINCPortSpeed tUeiARINCPortSpeed**
  ARINC port speed.
- typedef enum **_tUeiARINCPortParity tUeiARINCPortParity**
  ARINC port parity.
- typedef enum **_tUeiARINCPortFrameCountingMode tUeiARINCPortFrameCounting-Mode**
  ARINC port frame counting mode.
- typedef enum **_tUeiARINCMessageType tUeiARINCMessageType**
  ARINC message type.
- typedef enum **_tUeiARINCSchedulerType tUeiARINCSchedulerType**
  ARINC scheduler type.
- typedef enum **_tUeiMIL1553FilterType tUeiMIL1553FilterType**
  1553 filter setting
- typedef enum **_tUeiMIL1553PortCoupling tUeiMIL1553PortCoupling**
  1553 port coupling
- typedef enum **_tUeiMIL1553PortOpMode tUeiMIL1553PortOpMode**
  1553 port operating mode
- typedef enum **_tUeiMIL1553PortActiveBus tUeiMIL1553PortActiveBus**
  1553 port bus

- typedef enum **_tUeiMIL1553Endian tUeiMIL1553Endian**
  1553 endian (ARINC-708 model only)
- typedef enum **_tUeiMIL1553BCMajorFlags tUeiMIL1553BCMajorFlags**
  1553 major frame flags
- typedef enum **_tUeiMIL1553BCMinorFlags tUeiMIL1553BCMinorFlags**
  1553 minor frame flags
- typedef enum **_tUeiMIL1553CommandType tUeiMIL1553CommandType**
  1553 commands
- typedef enum **_tUeiMIL1553RTControlType tUeiMIL1553RTControlType**
  1553 control command
- typedef enum **_tUeiMIL1553BCRetryType tUeiMIL1553BCRetryType**
  1553 commands
- typedef enum **_tUeiMIL1553BCFrameType tUeiMIL1553BCFrameType**
  1553 BC frame type
- typedef enum **_tUeiMIL1553ModeCommandTypes tUeiMIL1553ModeCommandTypes**
  1553 mode command codes
- typedef enum **_tUeiMIL1553FrameType tUeiMIL1553FrameType**
  MIL-1553 frame type.
- typedef enum **_tUeiMIL1553BCOps tUeiMIL1553BCOps**
  1553 BC operations
- typedef enum **_tUeiMIL1553BCGotoOps tUeiMIL1553BCGotoOps**
  1553 BC GoTo command types
- typedef enum **_tUeiMIL1553A708Ops tUeiMIL1553A708Ops**
  ARINC-708 operations.
- typedef enum **_tUeiLogFileFormat tUeiLogFileFormat**
  Log file format.
- typedef enum **_tUeiCoupling tUeiCoupling**
  Capacitive coupling.
- typedef enum **_tUeiLVDTWiringScheme tUeiLVDTWiringScheme**
  LVDT/RVDT excitation wiring schemes.
- typedef enum **_tUeiQuadratureDecodingType tUeiQuadratureDecodingType**
  Quadrature decoding type.
- typedef enum **_tUeiQuadratureZeroIndexPhase tUeiQuadratureZeroIndexPhase**

Z input phase.

- typedef enum **_tUeiSynchroResolverMode tUeiSynchroResolverMode**
  Synchro/Resolver Mode.
- typedef enum **_tUeiSynchroMessageType tUeiSynchroMessageType**
  Synchro/Resolver message data types.
- typedef enum **_tUeiDOPWMMode tUeiDOPWMMode**
  Digital output PWM mode.
- typedef enum **_tUeiDOPWMOutputMode tUeiDOPWMOutputMode**
  Digital output PWM output mode.
- typedef enum **_tUeiFlushAction tUeiFlushAction**
  Flush actions.
- typedef enum **_tUeiCustomScaleType tUeiCustomScaleType**
  Custom Scale type.
- typedef enum **_tUeiIRIGTimeCodeFormat tUeiIRIGTimeCodeFormat**
  IRIG Time code format.
- typedef enum **_tUeiIRIGTimeKeeper1PPSSource tUeiIRIGTimeKeeper1PPSSource**
  IRIG time keeper 1 PPS signal source.
- typedef enum **_tUeiIRIGDecoderInputType tUeiIRIGDecoderInputType**
  Time decoder input.
- typedef enum **_tUeiIRIGTimeFormat tUeiIRIGTimeFormat**
  IRIG time format type.
- typedef enum **_tUeiIRIGDOTTLSource tUeiIRIGDOTTLSource**
  IRIG DO outputs source.
- typedef enum **_tUeiIRIGEventSource tUeiIRIGEventSource**
  IRIG event module source.
- typedef enum **_tUeiHDLCPortPhysical tUeiHDLCPortPhysical**
  HDLC port physical interface.
- typedef enum **_tUeiHDLCPortAbortSymbol tUeiHDLCPortAbortSymbol**
  HDLC port abort symbol.
- typedef enum **_tUeiHDLCPortUnderrunAction tUeiHDLCPortUnderrunAction**
  HDLC port underrun action.
- typedef enum **_tUeiHDLCPortEncoding tUeiHDLCPortEncoding**
  HDLC port encoding.

- typedef enum **_tUeiHDLCPortClockSource tUeiHDLCPortClockSource**
  HDLC port clock source.
- typedef enum **_tUeiHDLCPortCRCMode tUeiHDLCPortCRCMode**
  HDLC port CRC mode.
- typedef enum **_tUeiHDLCPortFilterMode tUeiHDLCPortFilterMode**
  HDLC port filter mode.
- typedef enum **_tUeiHDLCPortPreamble tUeiHDLCPortPreamble**
  HDLC port preamble pattern.
- typedef enum **_tUeiHDLCPortPreambleSize tUeiHDLCPortPreambleSize**
  HDLC port preamble size.
- typedef enum **_tUeiHDLCPortIdleCharacter tUeiHDLCPortIdleCharacter**
  HDLC port idle character.
- typedef enum **_tUeiWatchDogCommand tUeiWatchDogCommand**
  Watchdog command.
- typedef enum **_tUeiVRDataType tUeiVRDataType**
  VR data message type.
- typedef enum **_tUeiVRMode tUeiVRMode**
  Variable reluctance measurement mode.
- typedef enum **_tUeiVRZCMode tUeiVRZCMode**
  Variable reluctance zero-crossing mode.
- typedef enum **_tUeiVRAPTMode tUeiVRAPTMode**
  Variable reluctance adaptive peak threshold mode.
- typedef enum **_tUeiVRFIFOMode tUeiVRFIFOMode**
  Variable reluctance FIFO mode.
- typedef enum **_tUeiVRDigitalSource tUeiVRDigitalSource**
  Variable reluctance digital output source.
- typedef enum **_tUeiFeatureEnable tUeiFeatureEnable**
  Specifies whether a feature is manually or automatically turned on or off.
- typedef enum **_tUeiEEPROMArea tUeiEEPROMArea**
  EEPROM area.
- typedef enum **_tUeiSync1PPSMode tUeiSync1PPSMode**
  1PPS synchronization mode
- typedef enum **_tUeiSync1PPSSource tUeiSync1PPSSource**
  the source of the 1PPS sync clock

- typedef enum **_tUeiSyncLine tUeiSyncLine**
  Backplane synchonization line.
- typedef enum **_tUeiSync1PPSOutput tUeiSync1PPSOutput**
  the output of the 1PPS sync clock
- typedef enum **_tUeiSync1PPSDataType tUeiSync1PPSDataType**
  1PPS status message type
- typedef enum **_tUeiSync1PPSTriggerOperation tUeiSync1PPSTriggerOperation**
  1PPS action
- typedef enum **_tUeiCSDBDataType tUeiCSDBDataType**
  CSDB data message type.
- typedef enum **_tUeiPTPState tUeiPTPState**
  PTP states.
- typedef enum **_tUeiDMMMeasurementMode tUeiDMMMeasurementMode**
  DMM measurement modes.
- typedef enum **_tUeiMeasurementUnits tUeiMeasurementUnits**
  measurement units
- typedef enum **_tUeiDMMFIRCutoff tUeiDMMFIRCutoff**
  DMM FIR cutoff frequency modes.
- typedef enum **_tUeiDMMZeroCrossingMode tUeiDMMZeroCrossingMode**
  DMM zero crossing detection modes.
- typedef enum **_tUeiAOAuxCommand tUeiAOAuxCommand**
  Auxiliary AO command.
- typedef enum **_tUeiMuxSyncOutputMode tUeiMuxSyncOutputMode**
  MUX device synchronization output mode.
- typedef enum **_tUeiMUXMessageType tUeiMUXMessageType**
  MUX message type.
- typedef enum **_tUeiMuxVoltage tUeiMuxVoltage**
  MUX device voltage level.
- typedef enum **_tUeiMuxDmmMode tUeiMuxDmmMode**
  MUX to DMM output select.
- typedef enum **_tUeiI2CPortSpeed tUeiI2CPortSpeed**
  I2C port speed.
- typedef enum **_tUeiI2CTTLLevel tUeiI2CTTLLevel**

I2C TTL level.

- typedef enum **_tUeiI2CSlaveAddressWidth tUeiI2CSlaveAddressWidth**
  I2C slave addressing mode.
- typedef enum **_tUeiI2CMessageType tUeiI2CMessageType**
  I2C message data types.
- typedef enum **_tUeiI2CCommand tUeiI2CCommand**
  I2C commands.
- typedef enum **_tUeiI2CSlaveDataMode tUeiI2CSlaveDataMode**
  I2C slave data modes.
- typedef enum **_tUeiI2CLoopback tUeiI2CLoopback**
  I2C loopback modes.
- typedef enum **_tUeiI2CBusCode tUeiI2CBusCode**
  Bus condition codes.
- typedef enum **_tUeiInitParameter tUeiInitParameter**
  Init parameters.

Enumerations

- enum **_tUeiAIChannelInputMode** {
  UeiAIChannelInputModeDifferential,
  UeiAIChannelInputModeSingleEnded }
  AI input mode.
- enum **_tUeiAIChannelMuxPos** {
  UeiAIChannelMuxPosOff,
  UeiAIChannelMuxPosAlt5V,
  UeiAIChannelMuxPosIn5V }
  AI mux modes.
- enum **_tUeiAOWaveformType** {
  UeiAOWaveformTypeSine,
  UeiAOWaveformTypePulse,
  UeiAOWaveformTypeTriangle,
  UeiAOWaveformTypeSawtooth,
  UeiAOWaveformTypeCustom }
  AO waveform type.
- enum **_tUeiAOWaveformMode** {
  UeiAOWaveformModeDDS,
  UeiAOWaveformModePLL }

AO waveform mode.

- enum **_tUeiAOWaveformXform** {
  UeiAOWaveformXformNone,
  UeiAOWaveformXformMirror,
  UeiAOWaveformXformInvert,
  UeiAOWaveformXformMirrorAndInvert }
  AO waveform transform.

- enum **_tUeiAOWaveformOffsetClockSource** {
  UeiAOWaveformOffsetClockSourceDIO0,
  UeiAOWaveformOffsetClockSourceDIO1,
  UeiAOWaveformOffsetClockSourceDAC,
  UeiAOWaveformOffsetClockSourcePLL,
  UeiAOWaveformOffsetClockSourceSW }
  AO waveform offset DAC clock source.

- enum **_tUeiAOWaveformClockSource** {
  UeiAOWaveformClockSourceSYNC2,
  UeiAOWaveformClockSourceSYNC0,
  UeiAOWaveformClockSourceALT0,
  UeiAOWaveformClockSourceTMR,
  UeiAOWaveformClockSourceDIO0,
  UeiAOWaveformClockSourcePLL,
  UeiAOWaveformClockSourceSW }
  AO waveform main clock source.

- enum **_tUeiAOWaveformClockSync** {
  UeiAOWaveformClockRouteNone,
  UeiAOWaveformClockRouteDIO1ToTrgOut,
  UeiAOWaveformClockRouteDIO0ToTrgOut,
  UeiAOWaveformClockRoutePLLToTrgOut,
  UeiAOWaveformClockRoutePLLToSYNC2,
  UeiAOWaveformClockRoutePLLToSYNC0 }
  AO waveform clock synchronization.

- enum **_tUeiAOWaveformOffsetTriggerSource** {
  UeiAOWaveformOffsetTriggerSourceNone,
  UeiAOWaveformOffsetTriggerSourceSYNC3,
  UeiAOWaveformOffsetTriggerSourceSYNC1,
  UeiAOWaveformOffsetTriggerSourceALT0,
  UeiAOWaveformOffsetTriggerSourceDIO1,
  UeiAOWaveformOffsetTriggerSourceSW }

AO waveform offset DAC trigger source.

- enum **_tUeiAOWaveformTriggerSource** {
  UeiAOWaveformTriggerSourceNone,
  UeiAOWaveformTriggerSourceCH0,
  UeiAOWaveformTriggerSourceSYNC3,
  UeiAOWaveformTriggerSourceSYNC1,
  UeiAOWaveformTriggerSourceALT0,
  UeiAOWaveformTriggerSourceDIO1,
  UeiAOWaveformTriggerSourceSW }
  AO waveform main DAC trigger source.

- enum **_tUeiAOWaveformCommandType** {
  UeiAOWaveformCommandShape,
  UeiAOWaveformCommandSweep,
  UeiAOWaveformCommandAWF }
  AO waveform message type.

- enum **_tUeiAOWaveformSweepControl** {
  UeiAOWaveformSweepUpStart,
  UeiAOWaveformSweepDownStart,
  UeiAOWaveformSweepUpDownStart,
  UeiAOWaveformSweepDownUpStart,
  UeiAOWaveformSweepStop }
  AO waveform sweep control.

- enum **_tUeiAODACMode** {
  UeiAODACModeDisconnected,
  UeiAODACModeBothConnected,
  UeiAODACModeAConnected,
  UeiAODACModeBConnected }
  DAC mode.

- enum **_tUeiAODiagChannel** {
  UeiAODiagnosticCurrent,
  UeiAODiagnosticDACAVoltage,
  UeiAODiagnosticDACBVoltage,
  UeiAODiagnosticVoltage,
  UeiAODiagnosticTemperature,
  UeiAODiagnosticNone }
  AO Diagnostic ADC channel.

- enum **_tUeiCounterMode** {
  UeiCounterModeCountEvents,
  UeiCounterModeMeasurePulseWidth,
  UeiCounterModeMeasurePeriod,
  UeiCounterModeGeneratePulse,
  UeiCounterModeGeneratePulseTrain,
  UeiCounterModePulseWidthModulation,
  UeiCounterModeQuadratureEncoder,
  UeiCounterModeDirectionCounter,
  UeiCounterModeTimedPeriodMeasurement,
  UeiCounterModeBinCounting }
  Counter/Timer mode.

- enum **_tUeiCounterSource** {
  UeiCounterSourceClock,
  UeiCounterSourceClockDiv2,
  UeiCounterSourceInput,
  UeiCounterSourceCounter0Out,
  UeiCounterSourceCounter1Out,
  UeiCounterSourceCounter2Out }
  Counter/Timer source.

- enum **_tUeiCounterGate** {
  UeiCounterGateInternal,
  UeiCounterGateExternal }
  Counter/Timer gate.

- enum **_tUeiCounterGateMode** {
  UeiCounterGateModeContinuous,
  UeiCounterGateModeSingleShot }
  Counter/Timer gate mode.

- enum **_tUeiCounterCaptureTimebase** {
  UeiCounterCaptureTimebase1x,
  UeiCounterCaptureTimebase2x,
  UeiCounterCaptureTimebaseSync0,
  UeiCounterCaptureTimebaseSync1,
  UeiCounterCaptureTimebaseSync2,
  UeiCounterCaptureTimebaseSync3 }
  Counter/Timer measurement timebase.

- enum **_tUeiDigitalEdge** {
  UeiDigitalEdgeRising,
  UeiDigitalEdgeFalling,
  UeiDigitalEdgeBoth }
  Digital Edge.

- enum **_tUeiDigitalInputMux** {
  UeiDigitalInputMuxTriState,
  UeiDigitalInputMuxPullUp,
  UeiDigitalInputMuxDiag }
  Digital Input Mux.

- enum **_tUeiDigitalTermination** {
  UeiDigitalTerminationNone,
  UeiDigitalTerminationPullUp,
  UeiDigitalTerminationPullDown,
  UeiDigitalTerminationPullUpPullDown }
  Digital output termination.

- enum **_tUeiTimingDuration** {
  UeiTimingDurationContinuous,
  UeiTimingDurationSingleShot }
  Timing Duration.

- enum **_tUeiTimingMode** {
  UeiTimingModeSimpleIO,
  UeiTimingModeBufferedIO,
  UeiTimingModeChangeDetection,
  UeiTimingModeTimeSequencing,
  UeiTimingModeDirectDataMapping,
  UeiTimingModeMessagingIO,
  UeiTimingModeAsyncIO,
  UeiTimingModeAsyncVMapIO,
  UeiTimingModeVMapIO }
  Timing mode.

- enum **_tUeiTimingClockSource** {
  UeiTimingClockSourceInternal,
  UeiTimingClockSourceExternal,
  UeiTimingClockSourceContinuous,
  UeiTimingClockSourceSoftware,
  UeiTimingClockSourceSlave,
  UeiTimingClockSourceExternalDividedByCounter,
  UeiTimingClockSourceSignal }

Clock source.

- enum **_tUeiSessionType** {
  UeiSessionTypeAI,
  UeiSessionTypeAO,
  UeiSessionTypeDI,
  UeiSessionTypeDO,
  UeiSessionTypeCI,
  UeiSessionTypeCO,
  UeiSessionTypeInfo,
  UeiSessionTypeSerial,
  UeiSessionTypeSynchronousSerial,
  UeiSessionTypeCAN,
  UeiSessionTypeARINC,
  UeiSessionTypeMIL1553,
  UeiSessionTypeIRIG,
  UeiSessionTypeSync,
  UeiSessionTypeDILineLevel,
  UeiSessionTypeDOLineLevel,
  UeiSessionTypeVR,
  UeiSessionTypeCSDB,
  UeiSessionTypeSSI,
  UeiSessionTypeSPI,
  UeiSessionTypeMUX,
  UeiSessionTypeI2C,
  UeiSessionTypeDiagnostic }
  Session type.

- enum **_tUeiSessionState** {
  UeiSessionStateUnknown,
  UeiSessionStateReserved,
  UeiSessionStateConfigured,
  UeiSessionStateRunning,
  UeiSessionStateFinished }
  Session state.

- enum **_tUeiEvent** {
  UeiEventFrameDone,
  UeiEventBufferDone,
  UeiEventError,
  UeiEventDigitalIn,
  UeiEventSessionDone,
  UeiEventFIFOWatermark,
  UeiEventPeriodicTimer }

Asynchronous event.

- enum **_tUeiTriggerSource** {
  UeiTriggerSourceImmediate,
  UeiTriggerSourceSoftware,
  UeiTriggerSourceExternal,
  UeiTriggerSourceSignal,
  UeiTriggerSourceNext1PPS,
  UeiTriggerSourceSyncLine0,
  UeiTriggerSourceSyncLine1,
  UeiTriggerSourceSyncLine2,
  UeiTriggerSourceSyncLine3 }
  Trigger Source.

- enum **_tUeiTriggerCondition** {
  UeiTriggerConditionRising,
  UeiTriggerConditionFalling }
  Trigger Condition.

- enum **_tUeiTriggerAction** {
  UeiTriggerActionStartSession,
  UeiTriggerActionStopSession,
  UeiTriggerActionBuffer }
  Trigger Action.

- enum **_tUeiDataStreamRelativeTo** {
  UeiDataStreamRelativeToCurrentPosition,
  UeiDataStreamRelativeToMostRecentSample }
  Buffer access reference.

- enum **_tUeiThermocoupleType** {
  UeiThermocoupleTypeE,
  UeiThermocoupleTypeJ,
  UeiThermocoupleTypeK,
  UeiThermocoupleTypeR,
  UeiThermocoupleTypeS,
  UeiThermocoupleTypeT,
  UeiThermocoupleTypeB,
  UeiThermocoupleTypeN,
  UeiThermocoupleTypeC }
  Thermocouple types.

- enum **_tUeiColdJunctionCompensationType** {
  UeiCJCTypeBuiltIn,
  UeiCJCTypeConstant,
  UeiCJCTypeResource }
  Cold junction sensor type.

- enum **_tUeiTemperatureScale** {
  UeiTemperatureScaleCelsius,
  UeiTemperatureScaleFahrenheit,
  UeiTemperatureScaleKelvin,
  UeiTemperatureScaleRankine }
  Temperature scales.

- enum **_tUeiRTDType** {
  UeiRTDType3750,
  UeiRTDType3850,
  UeiRTDType3902,
  UeiRTDType3911,
  UeiRTDType3916,
  UeiRTDType3920,
  UeiRTDType3926,
  UeiRTDType3928,
  UeiRTDTypeCustom }
  RTD sensor type.

- enum **_tUeiReferenceResistorType** {
  UeiRefResistorBuiltIn,
  UeiRefResistorExternal }
  Reference resistor type.

- enum **_tUeiStrainGageBridgeType** {
  UeiStrainGageQuarterBridgeI,
  UeiStrainGageQuarterBridgeII,
  UeiStrainGageHalfBridgeI,
  UeiStrainGageHalfBridgeII,
  UeiStrainGageFullBridgeI,
  UeiStrainGageFullBridgeII,
  UeiStrainGageFullBridgeIII }
  Strain gage bridge configurations.

- enum **_tUeiSensorBridgeType** {
  UeiSensorQuarterBridge,
  UeiSensorHalfBridge,
  UeiSensorFullBridge,
  UeiSensorNoBridge }

Sensor bridge configurations.

- enum **_tUeiMeasurementType** {
  UeiMeasurementVoltage,
  UeiMeasurementThermocouple,
  UeiMeasurementStrainGage,
  UeiMeasurementVoltageWithExcitation,
  UeiMeasurementRTD,
  UeiMeasurementResistance,
  UeiMeasurementAccelerometer,
  UeiMeasurementLVDT,
  UeiMeasurementSynchroResolver,
  UeiMeasurementCurrent }
  Measurement type.

- enum **_tUeiOSType** {
  UeiOSWindows,
  UeiOSLinux,
  UeiOSPharlapEts,
  UeiOSMacOS }
  OS Type.

- enum **_tUeiSerialPortMode** {
  UeiSerialModeRS232,
  UeiSerialModeRS485HalfDuplex,
  UeiSerialModeRS485FullDuplex }
  Serial port mode.

- enum **_tUeiSerialPortSpeed** {
  UeiSerialBitsPerSecond110,
  UeiSerialBitsPerSecond300,
  UeiSerialBitsPerSecond600,
  UeiSerialBitsPerSecond1200,
  UeiSerialBitsPerSecond2400,
  UeiSerialBitsPerSecond4800,
  UeiSerialBitsPerSecond9600,
  UeiSerialBitsPerSecond14400,
  UeiSerialBitsPerSecond19200,
  UeiSerialBitsPerSecond28800,
  UeiSerialBitsPerSecond38400,
  UeiSerialBitsPerSecond57600,
  UeiSerialBitsPerSecond115200,
  UeiSerialBitsPerSecond128000,

UeiSerialBitsPerSecond250000,
UeiSerialBitsPerSecond256000,
UeiSerialBitsPerSecond1000000,
UeiSerialBitsPerSecondCustom }
Serial port speed.

- enum **_tUeiSerialPortDataBits** {
UeiSerialDataBits5,
UeiSerialDataBits6,
UeiSerialDataBits7,
UeiSerialDataBits8 }
Serial port number of data bits per character frame.

- enum **_tUeiSerialPortParity** {
UeiSerialParityNone,
UeiSerialParityOdd,
UeiSerialParityEven,
UeiSerialParityMark,
UeiSerialParitySpace }
Serial port parity.

- enum **_tUeiSerialPortStopBits** {
UeiSerialStopBits1,
UeiSerialStopBits1_5,
UeiSerialStopBits2 }
Serial port number of stop bits.

- enum **_tUeiSerialPortFlowControl** {
UeiSerialFlowControlNone,
UeiSerialFlowControlRtsCts,
UeiSerialFlowControlXonXoff }
Serial port flow control setting.

- enum **_tUeiSerialPortMinorFrameMode** {
UeiSerialMinorFrameModeFixedLength,
UeiSerialMinorFrameModeZeroChar,
UeiSerialMinorFrameModeVariableLength }
Minor frame delay mode.

- enum **_tUeiSerialReadDataType** {
UeiSerialReadDataTypeNoTimestamp = 0,
UeiSerialReadDataTypeTimestamp = 1 }
CUeiSerialReader data types.

- enum **_tUeiCANPortSpeed** {
  UeiCANBitsPerSecond10K,
  UeiCANBitsPerSecond20K,
  UeiCANBitsPerSecond50K,
  UeiCANBitsPerSecond100K,
  UeiCANBitsPerSecond125K,
  UeiCANBitsPerSecond250K,
  UeiCANBitsPerSecond500K,
  UeiCANBitsPerSecond800K,
  UeiCANBitsPerSecond1M }
  CAN port speed.

- enum **_tUeiCANFrameFormat** {
  UeiCANFrameBasic,
  UeiCANFrameExtended }
  CAN frame format.

- enum **_tUeiCANFrameType** {
  UeiCANFrameTypeData,
  UeiCANFrameTypeRemote,
  UeiCANFrameTypeError }
  CAN frame type.

- enum **_tUeiCANPortMode** {
  UeiCANPortModeNormal,
  UeiCANPortModePassive }
  CAN operation mode.

- enum **_tUeiWheatstoneBridgeBranch** {
  UeiWheatstoneBridgeR1,
  UeiWheatstoneBridgeR2,
  UeiWheatstoneBridgeR3,
  UeiWheatstoneBridgeR4 }
  Wheatstone bridge branches.

- enum **_tUeiSignalSource** {
  UeiSignalStartTrigger,
  UeiSignalStopTrigger,
  UeiSignalScanClock,
  UeiSignalConvertClock,
  UeiSignalBackplane,
  UeiSignalSoftware,
  UeiSignalExternal,
  UeiSignalPLL,
  UeiSignalSync1PPS }

The source of a synchronization signal.

- enum **_tUeiWiringScheme** {
  UeiTwoWires,
  UeiFourWires,
  UeiSixWires,
  UeiThreeWires }
  Sensor with excitation wiring schemes.

- enum **_tUeiARINCPortSpeed** {
  UeiARINCBitsPerSecond12500,
  UeiARINCBitsPerSecond100000 }
  ARINC port speed.

- enum **_tUeiARINCPortParity** {
  UeiARINCParityNone,
  UeiARINCParityOdd,
  UeiARINCParityEven }
  ARINC port parity.

- enum **_tUeiARINCPortFrameCountingMode** {
  UeiARINCFrameCountAll,
  UeiARINCFrameCountGood,
  UeiARINCFrameCountFIFO,
  UeiARINCFrameCountTrigger,
  UeiARINCFrameParityError }
  ARINC port frame counting mode.

- enum **_tUeiARINCMessageType** {
  UeiARINCMessageTypeWord,
  UeiARINCMessageTypeScheduledWord,
  UeiARINCMessageTypePauseScheduler,
  UeiARINCMessageTypeResumeScheduler,
  UeiARINCMessageTypeSetMajorMinorActivePage }
  ARINC message type.

- enum **_tUeiARINCSchedulerType** {
  UeiARINCSchedulerTypeNormal,
  UeiARINCSchedulerTypeFramed,
  UeiARINCSchedulerTypeMajorMinorFrame }
  ARINC scheduler type.

- enum **_tUeiMIL1553FilterType** {
  UeiMIL1553FilterByRt,
  UeiMIL1553FilterByRtSa,
  UeiMIL1553FilterByRtSaSize,
  UeiMIL1553FilterValidationEntry }
  1553 filter setting

- enum **_tUeiMIL1553PortCoupling** {
  UeiMIL1553CouplingDisconnected,
  UeiMIL1553CouplingTransformer,
  UeiMIL1553CouplingLocalStub,
  UeiMIL1553CouplingDirect }
  1553 port coupling

- enum **_tUeiMIL1553PortOpMode** {
  UeiMIL1553OpModeBusMonitor,
  UeiMIL1553OpModeRemoteTerminal,
  UeiMIL1553OpModeBusController,
  UeiMIL1553OpModeARINC708 }
  1553 port operating mode

- enum **_tUeiMIL1553PortActiveBus** {
  UeiMIL1553OpModeBusA,
  UeiMIL1553OpModeBusB,
  UeiMIL1553OpModeBusBoth }
  1553 port bus

- enum **_tUeiMIL1553Endian** {
  UeiMIL1553SmallEndian,
  UeiMIL1553BigEndian }
  1553 endian (ARINC-708 model only)

- enum **_tUeiMIL1553BCMajorFlags** {
  UeiMIL1553MjLink = (1L<<9),
  UeiMIL1553MjEnable = (1L<<8),
  UeiMIL1553MjExecuteOnce = (1L<<7),
  UeiMIL1553MjSendMessage = (1L<<6),
  UeiMIL1553MjSwapEnabled = (1L<<5),
  UeiMIL1553MjDoneOnce = (1L<<4) }
  1553 major frame flags

- enum **_tUeiMIL1553BCMinorFlags** {
  UeiMIL1553MnEnable = (1L<<8),
  UeiMIL1553MnExecuteOnce = (1L<<8)+(1L<<7),
  UeiMIL1553MnCurrentBusB = (1L<<6),
  UeiMIL1553MnRTRecovered = (1L<<5),
  UeiMIL1553MnNDoneOnce = (1L<<4),
  UeiMIL1553MnExecError = (1L<<3) }
  1553 minor frame flags

- enum **_tUeiMIL1553CommandType** {
  UeiMIL1553CmdBCRT,
  UeiMIL1553CmdRTBC,
  UeiMIL1553CmdRTRT,
  UeiMIL1553CmdModeTxNoData,
  UeiMIL1553CmdModeTxWithData,
  UeiMIL1553CmdModeRxWithData,
  UeiMIL1553CmdBCRTBroadcast,
  UeiMIL1553CmdRTRTBroadcast,
  UeiMIL1553CmdModeTxNoDataBroadcast,
  UeiMIL1553CmdModeRxWithDataBroadcast }
  1553 commands

- enum **_tUeiMIL1553RTControlType** {
  UeiMIL1553Disable,
  UeiMIL1553Enable,
  UeiMIL1553EnableMask,
  UeiMIL1553RTSetRtBcBlock,
  UeiMIL1553RTSetBcRtBlock,
  UeiMIL1553RTSetValid,
  UeiMIL1553RTResponseTiming }
  1553 control command

- enum **_tUeiMIL1553BCRetryType** {
  UeiMIL1553BCR_IRT = (1L<<14),
  UeiMIL1553BCR_RUS = (1L<<13),
  UeiMIL1553BCR_RUD = (1L<<12),
  UeiMIL1553BCR_RWB = (1L<<11),
  UeiMIL1553BCR_RIS = (1L<<10),
  UeiMIL1553BCR_RBB = (1L<<9),
  UeiMIL1553BCR_RTE = (1L<<8),
  UeiMIL1553BCR_RWC = (1L<<7),
  UeiMIL1553BCR_RE = (1L<<6),

UeiMIL1553BCR_RNR = (1L<<5),
UeiMIL1553BCR_ERE = (1L<<4),
UeiMIL1553BCR_ESR = (1L<<3) }
1553 commands

- enum **_tUeiMIL1553BCFrameType** {
UeiMIL1553BCFrameUndef,
UeiMIL1553BCFrameMajor,
UeiMIL1553BCFrameMinor }
1553 BC frame type

- enum **_tUeiMIL1553ModeCommandTypes** {
UeiMIL1553ModeDynamicBusControl = 0x0,
UeiMIL1553ModeSynchronize = 0x1,
UeiMIL1553ModeTransmitStatusWord = 0x2,
UeiMIL1553ModeStartSelfTest = 0x3,
UeiMIL1553ModeTxShutdown = 0x4,
UeiMIL1553ModeTxShutdownOver = 0x5,
UeiMIL1553ModeInhTerminalFlag = 0x6,
UeiMIL1553ModeInhTerminalFlagOver = 0x7,
UeiMIL1553ModeResetRt = 0x8,
UeiMIL1553ModeTxVectorWord = 0x10,
UeiMIL1553ModeSynchronizeData = 0x11,
UeiMIL1553ModeTxLastCommand = 0x12,
UeiMIL1553ModeTxBITWord = 0x13,
UeiMIL1553ModeSelectedTxShutdown = 0x14,
UeiMIL1553ModeOverrideTxShutdown = 0x15 }
1553 mode command codes

- enum **_tUeiMIL1553FrameType** {
UeiMIL1553FrameTypeRtData,
UeiMIL1553FrameTypeBusWriter,
UeiMIL1553FrameTypeBusMon,
UeiMIL1553FrameTypeBusMonCmd,
UeiMIL1553FrameTypeRtStatusData,
UeiMIL1553FrameTypeRtStatusLast,
UeiMIL1553FrameTypeRtParameters,
UeiMIL1553FrameTypeRtControlData,
UeiMIL1553FrameTypeError,
UeiMIL1553FrameTypeBCCBData,
UeiMIL1553FrameTypeBCCBStatus,
UeiMIL1553BCSchedFrame,

UeiMIL1553BCControlFrame,
UeiMIL1553BCStatusFrame,
UeiMIL1553FrameTypeA708Data,
UeiMIL1553FrameTypeA708Control,
UeiMIL1553FrameTypeGeneric = 100 }
MIL-1553 frame type.

- enum **_tUeiMIL1553BCOps** {
UeiMIL1553BcOpDisable,
UeiMIL1553BcOpEnable,
UeiMIL1553BcOpStepMj,
UeiMIL1553BcOpStepMn,
UeiMIL1553BcOpGoto,
UeiMIL1553BcOpMnSelect,
UeiMIL1553BcOpMjSwap }
1553 BC operations

- enum **_tUeiMIL1553BCGotoOps** {
UeiMIL1553GotoBcbNoret = 0,
UeiMIL1553GotoMnNoret,
UeiMIL1553GotoBcbBcb,
UeiMIL1553GotoMnBCB,
UeiMIL1553GotoBcbMn,
UeiMIL1553GotoMnMn }
1553 BC GoTo command types

- enum **_tUeiMIL1553A708Ops** {
UeiMIL1553A708OpDisable,
UeiMIL1553A708OpEnable,
UeiMIL1553A708FIFODisable,
UeiMIL1553A708FIFOEnable,
UeiMIL1553A708FIFOClear }
ARINC-708 operations.

- enum **_tUeiLogFileFormat** {
UeiLogFileCSV,
UeiLogFileBinary }
Log file format.

- enum **_tUeiCoupling** {
UeiCouplingDC,
UeiCouplingAC,
UeiCouplingAC_1Hz,
UeiCouplingAC_100mHz }

Capacitive coupling.

- enum **_tUeiLVDTWiringScheme** {
  UeiLVDTFourWires,
  UeiLVDTFiveWires }
  LVDT/RVDT excitation wiring schemes.

- enum **_tUeiQuadratureDecodingType** {
  UeiQuadratureDecodingType1x,
  UeiQuadratureDecodingType2x,
  UeiQuadratureDecodingType4x }
  Quadrature decoding type.

- enum **_tUeiQuadratureZeroIndexPhase** {
  UeiQuadratureZeroIndexPhaseZHigh,
  UeiQuadratureZeroIndexPhaseALowBLow,
  UeiQuadratureZeroIndexPhaseALowBHigh,
  UeiQuadratureZeroIndexPhaseAHighBLow,
  UeiQuadratureZeroIndexPhaseAHighBHigh }
  Z input phase.

- enum **_tUeiSynchroResolverMode** {
  UeiSynchroMode,
  UeiResolverMode,
  UeiSynchroZGroundMode }
  Synchro/Resolver Mode.

- enum **_tUeiSynchroMessageType** {
  UeiSynchroMessageStageAngles,
  UeiSynchroMessageUpdate }
  Synchro/Resolver message data types.

- enum **_tUeiDOPWMMode** {
  UeiDOPWMDisabled,
  UeiDOPWMSoftStart,
  UeiDOPWMSoftStop,
  UeiDOPWMSoftBoth,
  UeiDOPWMContinuous,
  UeiDOPWMGated }
  Digital output PWM mode.

- enum **_tUeiDOPWMOutputMode** {
  UeiDOPWMOutputPush,
  UeiDOPWMOutputPull,
  UeiDOPWMOutputPushPull,
  UeiDOPWMOutputOff }

Digital output PWM output mode.

- enum **_tUeiFlushAction** {
  UeiFlushReadBuffer = 0x01,
  UeiFlushDiscardReadBuffer = 0x02,
  UeiFlushWriteBuffer = 0x04,
  UeiFlushDiscardWriteBuffer = 0x08 }
  Flush actions.

- enum **_tUeiCustomScaleType** {
  UeiCustomScaleNone,
  UeiCustomScaleLinear,
  UeiCustomScalePolynomial,
  UeiCustomScaleTable }
  Custom Scale type.

- enum **_tUeiIRIGTimeCodeFormat** {
  UeiIRIGTimeCodeFormatA,
  UeiIRIGTimeCodeFormatB,
  UeiIRIGTimeCodeFormatD_100Hz,
  UeiIRIGTimeCodeFormatD_1000Hz,
  UeiIRIGTimeCodeFormatE_100Hz,
  UeiIRIGTimeCodeFormatE_1000Hz,
  UeiIRIGTimeCodeFormatG,
  UeiIRIGTimeCodeFormatH_100Hz,
  UeiIRIGTimeCodeFormatH_1000Hz }
  IRIG Time code format.

- enum **_tUeiIRIGTimeKeeper1PPSSource** {
  UeiIRIG1PPSInternal,
  UeiIRIG1PPSInputTimeCode,
  UeiIRIG1PPSGPS,
  UeiIRIG1PPSRFIn,
  UeiIRIG1PPSExternalTTL0,
  UeiIRIG1PPSExternalTTL1,
  UeiIRIG1PPSExternalTTL2,
  UeiIRIG1PPSExternalTTL3,
  UeiIRIG1PPSExternalSync0,
  UeiIRIG1PPSExternalSync1,
  UeiIRIG1PPSExternalSync2,
  UeiIRIG1PPSExternalSync3 }
  IRIG time keeper 1 PPS signal source.

- enum **_tUeiIRIGDecoderInputType** {
  UeiIRIGDecoderInputAM,
  UeiIRIGDecoderInputManchesterRF0,
  UeiIRIGDecoderInputManchesterRF1,
  UeiIRIGDecoderInputManchesterIO0,
  UeiIRIGDecoderInputManchesterIO1,
  UeiIRIGDecoderInputNRZRF0,
  UeiIRIGDecoderInputNRZRF1,
  UeiIRIGDecoderInputNRZIO0,
  UeiIRIGDecoderInputNRZIO1,
  UeiIRIGDecoderInputGPS }
  Time decoder input.

- enum **_tUeiIRIGTimeFormat** {
  UeiIRIGBCDTime,
  UeiIRIGSBSTime,
  UeiIRIGANSITime,
  UeiIRIGGPSTime,
  UeiIRIGTREG,
  UeiIRIGTimeCode,
  UeiIRIGSecondsSinceUnixEpochTime }
  IRIG time format type.

- enum **_tUeiIRIGDOTTLSource** {
  UeiIRIGDOTTLAMtoNRZ,
  UeiIRIGDOTTLGPSFixValid,
  UeiIRIGDOTTLGPSAntennaShorted,
  UeiIRIGDOTTLGPSAntennaOK,
  UeiIRIGDOTTLGPSTxD1,
  UeiIRIGDOTTLGPSTxD0,
  UeiIRIGDOTTLManchesterIItoNRZ,
  UeiIRIGDOTTLSYNC3,
  UeiIRIGDOTTLSYNC2,
  UeiIRIGDOTTLSYNC1,
  UeiIRIGDOTTLSYNC0,
  UeiIRIGDOTTLOutputCarrierFrequency,
  UeiIRIGDOTTLPLLFrequency,
  UeiIRIGDOTTLPrecision10MHZ,
  UeiIRIGDOTTLPrecision5MHZ,
  UeiIRIGDOTTLPrecision1MHZ,
  UeiIRIGDOTTLNRZStartStrobe,
  UeiIRIGDOTTLManchesterIITimeCode,

UeiIRIGDOTTLNRZTimeCode,
UeiIRIGDOTTLGPS1PPS,
UeiIRIGDOTTL1PPH,
UeiIRIGDOTTL1PPM,
UeiIRIGDOTTL1PPS,
UeiIRIGDOTTL0_1S,
UeiIRIGDOTTL0_01S,
UeiIRIGDOTTL1uS,
UeiIRIGDOTTLLogic1,
UeiIRIGDOTTLLogic0 ,
UeiIRIGDOTTLEventChannel1,
UeiIRIGDOTTLEventChannel2,
UeiIRIGDOTTLEventChannel3 }
IRIG DO outputs source.

- enum **_tUeiIRIGEventSource** { ,
UeiDaqIRIGEventSYNC1,
UeiDaqIRIGEventSYNC2,
UeiDaqIRIGEventSYNC3,
UeiDaqIRIGEvent1PPS,
UeiDaqIRIGEventEXTT,
UeiDaqIRIGEventEINV,
UeiDaqIRIGEventTTL0,
UeiDaqIRIGEventTTL1,
UeiDaqIRIGEventTTL2,
UeiDaqIRIGEventTTL3 }
IRIG event module source.

- enum **_tUeiHDLCPortPhysical** {
UeiHDLCPortRS232,
UeiHDLCPortRS485,
UeiHDLCPortRS422,
UeiHDLCPortV35 }
HDLC port physical interface.

- enum **_tUeiHDLCPortAbortSymbol** {
UeiHDLCPortAbort7,
UeiHDLCPortAbort15 }
HDLC port abort symbol.

- enum **_tUeiHDLCPortUnderrunAction** {
UeiHDLCPortUnderrunFinish,
UeiHDLCPortUnderrunFlags }

HDLC port underrun action.

- enum **_tUeiHDLCPortEncoding** {
  UeiHDLCPortEncodingNRZ,
  UeiHDLCPortEncodingNRZB,
  UeiHDLCPortEncodingNRZI,
  UeiHDLCPortEncodingNRZIMark,
  UeiHDLCPortEncodingNRZISpace,
  UeiHDLCPortEncodingBiphaseMark,
  UeiHDLCPortEncodingBiphaseSpace,
  UeiHDLCPortEncodingBiphaseLevel,
  UeiHDLCPortEncodingBiphaseDiff }
  HDLC port encoding.

- enum **_tUeiHDLCPortClockSource** {
  UeiHDLCPortClockExternalPin,
  UeiHDLCPortClockBRG,
  UeiHDLCPortClockDPLL,
  UeiHDLCPortClockDPLLDiv8,
  UeiHDLCPortClockDPLLDiv16,
  UeiHDLCPortClockDPLLSRCBRG }
  HDLC port clock source.

- enum **_tUeiHDLCPortCRCMode** {
  UeiHDLCPortCRCNone,
  UeiHDLCPortCRCUser,
  UeiHDLCPortCRC16CCITT,
  UeiHDLCPortCRC16,
  UeiHDLCPortCRC32 }
  HDLC port CRC mode.

- enum **_tUeiHDLCPortFilterMode** {
  UeiHDLCPortFilterNone,
  UeiHDLCPortFilterA16,
  UeiHDLCPortFilterA24,
  UeiHDLCPortFilterA32,
  UeiHDLCPortFilterEALS,
  UeiHDLCPortFilterEA24,
  UeiHDLCPortFilterEAMS,
  UeiHDLCPortFilterEAMS16 }
  HDLC port filter mode.

- enum **_tUeiHDLCPortPreamble** {
  UeiHDLCPortPreambleNone,
  UeiHDLCPortPreambleZero,
  UeiHDLCPortPreambleOne,
  UeiHDLCPortPreambleFlag,
  UeiHDLCPortPreamble10,
  UeiHDLCPortPreamble01 }
  HDLC port preamble pattern.

- enum **_tUeiHDLCPortPreambleSize** {
  UeiHDLCPortPreambleSize16,
  UeiHDLCPortPreambleSize32,
  UeiHDLCPortPreambleSize64 }
  HDLC port preamble size.

- enum **_tUeiHDLCPortIdleCharacter** {
  UeiHDLCPortIdleFlag,
  UeiHDLCPortIdleZero,
  UeiHDLCPortIdleOne,
  UeiHDLCPortIdleMark,
  UeiHDLCPortIdleSpace,
  UeiHDLCPortIdleMS,
  UeiHDLCPortIdle01 }
  HDLC port idle character.

- enum **_tUeiWatchDogCommand** {
  UeiWatchDogDisable,
  UeiWatchDogEnableClearOnReceive,
  UeiWatchDogEnableClearOnTransmit,
  UeiWatchDogEnableClearOnCommand,
  UeiWatchDogClearTimer }
  Watchdog command.

- enum **_tUeiVRDataType** {
  UeiVRDataPositionVelocity,
  UeiVRDataFifoPosition,
  UeiVRDataADCFifo,
  UeiVRDataADCStatus }
  VR data message type.

- enum **_tUeiVRMode** {
  UeiVRModeDecoder,
  UeiVRModeCounterTimed,
  UeiVRModeCounterNPulses,
  UeiVRModeCounterZPulse }

Variable reluctance measurement mode.

- enum **_tUeiVRZCMode** {
  UeiZCModeChip,
  UeiZCModeLogic,
  UeiZCModeFixed }
  Variable reluctance zero-crossing mode.

- enum **_tUeiVRAPTMode** {
  UeiAPTModeChip,
  UeiAPTModeLogic,
  UeiAPTModeFixed,
  UeiAPTModeTTL }
  Variable reluctance adaptive peak threshold mode.

- enum **_tUeiVRFIFOMode** {
  UeiFIFOModeDisabled,
  UeiFIFOModePosition,
  UeiFIFOModePosAndTS }
  Variable reluctance FIFO mode.

- enum **_tUeiVRDigitalSource** {
  UeiVRDigitalSourceDisable,
  UeiVRDigitalSourceForceHigh,
  UeiVRDigitalSourceForceLow,
  UeiVRDigitalSourceZTooth7,
  UeiVRDigitalSourceZTooth6,
  UeiVRDigitalSourceZTooth5,
  UeiVRDigitalSourceZTooth4,
  UeiVRDigitalSourceZTooth3,
  UeiVRDigitalSourceZTooth2,
  UeiVRDigitalSourceZTooth1,
  UeiVRDigitalSourceZTooth0,
  UeiVRDigitalSourceNPulse7,
  UeiVRDigitalSourceNPulse6,
  UeiVRDigitalSourceNPulse5,
  UeiVRDigitalSourceNPulse4,
  UeiVRDigitalSourceNPulse3,
  UeiVRDigitalSourceNPulse2,
  UeiVRDigitalSourceNPulse1,
  UeiVRDigitalSourceNPulse0 }
  Variable reluctance digital output source.

- enum **_tUeiFeatureEnable** {
  UeiFeatureDisabled,
  UeiFeatureEnabled,
  UeiFeatureAuto }
  Specifies whether a feature is manually or automatically turned on or off.

- enum **_tUeiEEPROMArea** {
  UeiEEPROMAreaCommon,
  UeiEEPROMAreaCalibration,
  UeiEEPROMAreaInitialization,
  UeiEEPROMAreaOperation,
  UeiEEPROMAreaShutdown,
  UeiEEPROMAreaNames,
  UeiEEPROMAreaWhole,
  UeiEEPROMAreaFlags }
  EEPROM area.

- enum **_tUeiSync1PPSMode** {
  UeiSyncClock,
  UeiSync1588,
  UeiSyncNTP,
  UeiSyncIRIG }
  1PPS synchronization mode

- enum **_tUeiSync1PPSSource** {
  UeiSync1PPSInternal,
  UeiSync1PPSInput0,
  UeiSync1PPSInput1 }
  the source of the 1PPS sync clock

- enum **_tUeiSyncLine** {
  UeiSyncLine0,
  UeiSyncLine1,
  UeiSyncLine2,
  UeiSyncLine3,
  UeiSyncNone }
  Backplane synchonization line.

- enum **_tUeiSync1PPSOutput** {
  UeiSync1PPSNone,
  UeiSync1PPSOutput0,
  UeiSync1PPSOutput1 }
  the output of the 1PPS sync clock

- enum **_tUeiSync1PPSDataType** {
  UeiSync1PPSDataFull,
  UeiSync1PPSDataLocked,
  UeiSync1PPSPTPStatus,
  UeiSync1PPSPTPUTCTime }
  1PPS status message type

- enum **_tUeiSync1PPSTriggerOperation** {
  UeiSync1PPSTriggerOnNextPPS,
  UeiSync1PPSTriggerOnNextPPSBroadCast }
  1PPS action

- enum **_tUeiCSDBDataType** {
  UeiCSDBDataFrame,
  UeiCSDBDataMessageByIndex,
  UeiCSDBDataMessageByAddress }
  CSDB data message type.

- enum **_tUeiPTPState**
  PTP states.

- enum **_tUeiDMMMeasurementMode**
  DMM measurement modes.

- enum **_tUeiMeasurementUnits**
  measurement units

- enum **_tUeiDMMFIRCutoff** {
  UeiDMMFIROff,
  UeiDMMFIR24Hz,
  UeiDMMFIR50Hz,
  UeiDMMFIR100Hz,
  UeiDMMFIR1kHz }
  DMM FIR cutoff frequency modes.

- enum **_tUeiDMMZeroCrossingMode** {
  UeiDMMZeroCrossingModeLevel,
  UeiDMMZeroCrossingModeMinMax,
  UeiDMMZeroCrossingModeRMS,
  UeiDMMZeroCrossingModeDC }
  DMM zero crossing detection modes.

- enum **_tUeiAOAuxCommand** {
  UeiAOAuxResetCircuitBreaker,
  UeiAOAuxShortOpenCircuit }

Auxiliary AO command.

- enum **_tUeiMuxSyncOutputMode** {
  UeiMuxSyncOutputLogic0,
  UeiMuxSyncOutputLogic1,
  UeiMuxSyncOutputSyncLine0,
  UeiMuxSyncOutputSyncLine1,
  UeiMuxSyncOutputSyncLine2,
  UeiMuxSyncOutputSyncLine3,
  UeiMuxSyncOutputRelaysReadyPulse0,
  UeiMuxSyncOutputRelaysReadyPulse1,
  UeiMuxSyncOutputRelaysReadyLogic0,
  UeiMuxSyncOutputRelaysReadyLogic1 }
  MUX device synchronization output mode.

- enum **_tUeiMUXMessageType**
  MUX message type.

- enum **_tUeiMuxVoltage** {
  UeiMuxVoltage2_42,
  UeiMuxVoltage3_3,
  UeiMuxVoltage4_2,
  UeiMuxVoltage5_1 }
  MUX device voltage level.

- enum **_tUeiMuxDmmMode**
  MUX to DMM output select.

- enum **_tUeiI2CPortSpeed** {
  UeiI2CBitsPerSecond100K,
  UeiI2CBitsPerSecond400K,
  UeiI2CBitsPerSecond1M,
  UeiI2CBitsPerSecondCustom }
  I2C port speed.

- enum **_tUeiI2CTTLLevel** {
  UeiI2CTTLLevel3_3V,
  UeiI2CTTLLevel5V }
  I2C TTL level.

- enum **_tUeiI2CSlaveAddressWidth** {
  UeiI2CSlaveAddress7bit,
  UeiI2CSlaveAddress10bit }
  I2C slave addressing mode.

- enum **_tUeiI2CMessageType** {
  UeiI2CMessageSlaveWrite,
  UeiI2CMessageMasterWrite,
  UeiI2CMessageSlaveRead,
  UeiI2CMessageMasterRead,
  UeiI2CMessageMasterAvailInput,
  UeiI2CMessageSlaveAvailInput,
  UeiI2CMessageSlaveAvailOutput }
  I2C message data types.

- enum **_tUeiI2CCommand** {
  UeiI2CCommandWrite,
  UeiI2CCommandRead,
  UeiI2CCommandWriteRead }
  I2C commands.

- enum **_tUeiI2CSlaveDataMode** {
  UeiI2CSlaveDataModeFIFO,
  UeiI2CSlaveDataModeRegister }
  I2C slave data modes.

- enum **_tUeiI2CLoopback** {
  UeiI2CLoopbackNone,
  UeiI2CLoopbackRelay,
  UeiI2CLoopbackFPGA }
  I2C loopback modes.

- enum **_tUeiI2CBusCode** {
  UeiI2CBusUnknown,
  UeiI2CBusStart,
  UeiI2CBusRestart,
  UeiI2CBusStop,
  UeiI2CBusAddressAck,
  UeiI2CBusAddressNack,
  UeiI2CBusDataAck,
  UeiI2CBusDataNack,
  UeiI2CBusAllData,
  UeiI2CBusClockStretchError }
  Bus condition codes.

- enum **_tUeiInitParameter** {
  UeiInitParameterModelId,
  UeiInitParameterModelOption,
  UeiInitParameterIOMSerial,
  UeiInitParameterIOMMfgDate,
  UeiInitParameterIOMBaseFrequency,
  UeiInitParameterTickSize,
  UeiInitParameterPeriod,
  UeiInitParameterWatchdogTimer,
  UeiInitParameterTime,
  UeiInitParameterOptions,
  UeiInitParameterComDelay,
  UeiInitParameterFWCT,
  UeiInitParameterLayerId,
  UeiInitParameterLayerOption,
  UeiInitParameterLayerSerial,
  UeiInitParameterEEPROMSize,
  UeiInitParameterLayerMfgDate,
  UeiInitParameterLayerCalDate,
  UeiInitParameterLayerCalExpiration,
  UeiInitParameterDQRev,
  UeiInitParameterFWRev }
  Init parameters.

3.1.1 Detailed Description

Typedefs and constants use thoughout the Ueidaq Framework

3.1.2 Typedef Documentation

3.1.2.1 typedef enum _tUeiAIChannelInputMode tUeiAIChannelInputMode

Input modes for Analog input channels

3.1.2.2 typedef enum _tUeiAIChannelMuxPos tUeiAIChannelMuxPos

Mux modes for AI self-test

3.1.2.3 typedef enum _tUeiAOAuxCommand tUeiAOAuxCommand

AO sessions use those commands to control auxiliary features such as circuit breakers, short or open circuit simulations

3.1.2.4 typedef enum _tUeiAODACMode tUeiAODACMode

Mode of the DAC connected to channel

3.1.2.5 typedef enum _tUeiAODiagChannel tUeiAODiagChannel

The available diagnostic measurements for each AO channel

3.1.2.6 typedef enum _tUeiAOWaveformClockSource tUeiAOWaveformClockSource

Clock used to pace main DAC

3.1.2.7 typedef enum _tUeiAOWaveformClockSync tUeiAOWaveformClockSync

Specifies where a clock signal should be routed to synchronize with other channels and/or layers Route signal to TrgOut to synchronize multiple channels on the same AO-364 Route signal to Sync lines to synchronize multiple AO-364s Only valid for channel 0 on AO-364

3.1.2.8 typedef enum _tUeiAOWaveformCommandType tUeiAOWaveformCommandType

AO waveform parameters are changed on the fly by sending commands to the AO session

3.1.2.9 typedef enum _tUeiAOWaveformMode tUeiAOWaveformMode

Timing mode used to generate waveform.

3.1.2.10 typedef enum _tUeiAOWaveformOffsetClockSource tUeiAOWaveformOffsetClockSource

Clock used to pace offset DAC

3.1.2.11 typedef enum _tUeiAOWaveformOffsetTriggerSource tUeiAOWaveformOffsetTriggerSource

source used to trigger a new period out of the offset DAC

3.1.2.12 typedef enum _tUeiAOWaveformSweepControl tUeiAOWaveformSweepControl

Command used to control sweep operation

3.1.2.13 typedef enum _tUeiAOWaveformTriggerSource tUeiAOWaveformTriggerSource

source used to trigger a new period out of the main DAC

3.1.2.14 typedef enum _tUeiAOWaveformType tUeiAOWaveformType

Shape of the waveform to program on a waveform generation capable device

3.1.2.15 typedef enum _tUeiAOWaveformXform tUeiAOWaveformXform

Transform applied to each waveform period

3.1.2.16 typedef enum _tUeiARINCMessageType tUeiARINCMessageType

Specifies the type of message sent or received by the ARINC stream

3.1.2.17 typedef enum _tUeiARINCPortFrameCountingMode tUeiARINCPortFrameCountingMode

ARINC port frame counting mode

3.1.2.18 typedef enum _tUeiARINCPortParity tUeiARINCPortParity

ARINC port parity

3.1.2.19 typedef enum _tUeiARINCPortSpeed tUeiARINCPortSpeed

ARINC port speed

3.1.2.20 typedef enum _tUeiARINCSchedulerType tUeiARINCSchedulerType

Specifies the type of scheduler used to emit scheduled words out of a given ARINC-429 port

3.1.2.21 typedef enum _tUeiCANFrameFormat tUeiCANFrameFormat

CAN frame format

3.1.2.22 typedef enum _tUeiCANFrameType tUeiCANFrameType

Specifies the type of CAN frame sent or received

3.1.2.23 typedef enum _tUeiCANPortMode tUeiCANPortMode

CAN operation mode

3.1.2.24 typedef enum _tUeiCANPortSpeed tUeiCANPortSpeed

CAN port speed

3.1.2.25 typedef enum _tUeiColdJunctionCompensationType tUeiColdJunction-CompensationType

Some terminal block connectors such as the DNA-STP-AIU include a built-in cold junction sensor. You can also provide your own.

3.1.2.26 typedef enum _tUeiCounterCaptureTimebase tUeiCounterCaptureTimebase

timebase used measure period, frequency and pulse width

3.1.2.27 typedef enum _tUeiCounterGate tUeiCounterGate

source of the gating signal

3.1.2.28 typedef enum _tUeiCounterGateMode tUeiCounterGateMode

gating mode

3.1.2.29 typedef enum _tUeiCounterMode tUeiCounterMode

Counter/Timer mode

3.1.2.30 typedef enum _tUeiCounterSource tUeiCounterSource

Source of the clock signal or the signal to count

3.1.2.31 typedef enum _tUeiCoupling tUeiCoupling

The type of coupling used to connect a signal to an analog input device

3.1.2.32 typedef enum _tUeiCSDBDataType tUeiCSDBDataType

CSDB sessions can read/write entire frame or single message block

3.1.2.33 typedef enum _tUeiCustomScaleType tUeiCustomScaleType

The type of cutom scale to apply to a scaled input channel

3.1.2.34 typedef enum _tUeiDataStreamRelativeTo tUeiDataStreamRelativeTo

Specifies what position to use as a reference in the framework's internal buffer

3.1.2.35 typedef enum _tUeiDigitalEdge tUeiDigitalEdge

Edge of a digital pulse

3.1.2.36 typedef enum _tUeiDigitalInputMux tUeiDigitalInputMux

Mode of the digital input mux installed on Industrial digital input devices.

3.1.2.37 typedef enum _tUeiDigitalTermination tUeiDigitalTermination

specified termination resistor applied to a digital output

3.1.2.38 typedef enum _tUeiDMMMeasurementMode tUeiDMMMeasurementMode

the modes of measurement for a DMM device

3.1.2.39 typedef enum _tUeiDOPWMMode tUeiDOPWMMode

Specifies the PWM mode for DO channels that support the capability With PWM mode you can replace the rising and falling edges with a pulse train to allow for soft start and/or stop

3.1.2.40 typedef enum _tUeiDOPWMOutputMode tUeiDOPWMOutputMode

specifies if PWM is enabled on low size, high size or both sides of the FET This setting is only valid for digital outputlines based on FET

3.1.2.41 typedef enum _tUeiEEPROMArea tUeiEEPROMArea

Specifies the area to read/write in EEPROM

3.1.2.42 typedef enum _tUeiEvent tUeiEvent

Asynchronous event

Examples:

AnalogInBufferedAsync.cpp, and **AnalogOutBufferedAsync.cpp**.

3.1.2.43 typedef enum _tUeiFeatureEnable tUeiFeatureEnable

Manual mode lets you turn on or off a feature from your code Automatic mode controls the feature from a setting saved in EEPROM

3.1.2.44 typedef enum _tUeiFlushAction tUeiFlushAction

Specifies how to flush a messaging port (serial, CAN, ARINC, MIL-1553)

3.1.2.45 typedef enum _tUeiHDLCPortAbortSymbol tUeiHDLCPortAbortSymbol

Symbol used to abort HDLC transmission

3.1.2.46 typedef enum _tUeiHDLCPortClockSource tUeiHDLCPortClockSource

clock source used to synchronize transmit and or receive lines

3.1.2.47 typedef enum _tUeiHDLCPortCRCMode tUeiHDLCPortCRCMode

Algorithm used to calculate the CRC

3.1.2.48 typedef enum _tUeiHDLCPortEncoding tUeiHDLCPortEncoding

HDLC port encoding

3.1.2.49 typedef enum _tUeiHDLCPortFilterMode tUeiHDLCPortFilterMode

filter mode

3.1.2.50 typedef enum _tUeiHDLCPortIdleCharacter tUeiHDLCPortIdleCharacter

HDLC port idle character

3.1.2.51 typedef enum _tUeiHDLCPortPhysical tUeiHDLCPortPhysical

HDLC port physical interface

3.1.2.52 typedef enum _tUeiHDLCPortPreamble tUeiHDLCPortPreamble

HDLC port preamble pattern

3.1.2.53 typedef enum _tUeiHDLCPortPreambleSize tUeiHDLCPortPreambleSize

HDLC port preamble size

3.1.2.54 typedef enum _tUeiHDLCPortUnderrunAction tUeiHDLCPortUnderrunAction

Action to take when underrun condition is detected

3.1.2.55 typedef enum _tUeiI2CBusCode tUeiI2CBusCode

Bus condition codes

3.1.2.56 typedef enum _tUeiI2CCommand tUeiI2CCommand

I2C commands

3.1.2.57 typedef enum _tUeiI2CLoopback tUeiI2CLoopback

I2C loopback modes

3.1.2.58 typedef enum _tUeiI2CMessageType tUeiI2CMessageType

I2C message data types

3.1.2.59 typedef enum _tUeiI2CSlaveDataMode tUeiI2CSlaveDataMode

I2C slave data modes

3.1.2.60 typedef enum _tUeiInitParameter tUeiInitParameter

Specifies the parameter to read/write

3.1.2.61 typedef enum _tUeiIRIGDecoderInputType tUeiIRIGDecoderInputType

The type of signal connected to time decoder input

3.1.2.62 typedef enum _tUeiIRIGDOTTLSource tUeiIRIGDOTTLSource

Specifies the source that drives an IRIG TTL output

3.1.2.63 typedef enum _tUeiIRIGEventSource tUeiIRIGEventSource

Specifies the source that drives the event module

3.1.2.64 typedef enum _tUeiIRIGTimeCodeFormat tUeiIRIGTimeCodeFormat

The time code format used by the time coder and the time decoder

3.1.2.65 typedef enum _tUeiIRIGTimeFormat tUeiIRIGTimeFormat

Specifies the format used to send/receive time to/from the IRIG device

3.1.2.66 typedef enum _tUeiIRIGTimeKeeper1PPSSource tUeiIRIGTimeKeeper1PPSSource

The source of the incoming 1 PPS signal used to keep time

3.1.2.67 typedef enum _tUeiLogFileFormat tUeiLogFileFormat

The format for logging acquired data to disk

3.1.2.68 typedef enum _tUeiLVDTWiringScheme tUeiLVDTWiringScheme

LVDT/RVDT excitation wiring schemes

3.1.2.69 typedef enum _tUeiMeasurementType tUeiMeasurementType

Measurement type

3.1.2.70 typedef enum _tUeiMeasurementUnits tUeiMeasurementUnits

the units of measurement for a device

3.1.2.71 typedef enum _tUeiMIL1553A708Ops tUeiMIL1553A708Ops

ARINC-708 operations

3.1.2.72 typedef enum _tUeiMIL1553BCFrameType tUeiMIL1553BCFrameType

MIL-1553 BC frame type

3.1.2.73 typedef enum _tUeiMIL1553BCGotoOps tUeiMIL1553BCGotoOps

1553 BC GoTo command types

3.1.2.74 typedef enum _tUeiMIL1553BCMajorFlags tUeiMIL1553BCMajorFlags

MIL-1553 BC major frame flags

3.1.2.75 typedef enum _tUeiMIL1553BCMinorFlags tUeiMIL1553BCMinorFlags

MIL-1553 BC minor frame flags

3.1.2.76 typedef enum _tUeiMIL1553BCOps tUeiMIL1553BCOps

1553 BC operations

3.1.2.77 typedef enum _tUeiMIL1553BCRetryType tUeiMIL1553BCRetryType

STD-MIL-1553 commands

3.1.2.78 typedef enum _tUeiMIL1553CommandType tUeiMIL1553CommandType

STD-MIL-1553 commands

3.1.2.79 typedef enum _tUeiMIL1553Endian tUeiMIL1553Endian

1553 port endian

3.1.2.80 typedef enum _tUeiMIL1553FilterType tUeiMIL1553FilterType

1553 filter settings

3.1.2.81 typedef enum _tUeiMIL1553FrameType tUeiMIL1553FrameType

Specifies the type of 1553 frame sent or received

3.1.2.82 typedef enum _tUeiMIL1553ModeCommandTypes tUeiMIL1553ModeCommandTypes

STD-MIL-1553 mode command codes. These mode codes are encoded in the Word Count field

3.1.2.83 typedef enum _tUeiMIL1553PortActiveBus tUeiMIL1553PortActiveBus

1553 port active bus

3.1.2.84 typedef enum _tUeiMIL1553PortCoupling tUeiMIL1553PortCoupling

1553 port coupling

3.1.2.85 typedef enum _tUeiMIL1553PortOpMode tUeiMIL1553PortOpMode

1553 port operating mode

3.1.2.86 typedef enum _tUeiMIL1553RTControlType tUeiMIL1553RTControlType

STD-MIL-1553 control commands

3.1.2.87 typedef enum _tUeiMuxDmmMode tUeiMuxDmmMode

MUX output to DMM modes

3.1.2.88 typedef enum _tUeiMUXMessageType tUeiMUXMessageType

MUX message type

3.1.2.89 typedef enum _tUeiMuxSyncOutputMode tUeiMuxSyncOutputMode

MUX devices can assert a synchronization output line when relays are configured

3.1.2.90 typedef enum _tUeiMuxVoltage tUeiMuxVoltage

Voltage levels for DC/DC during relay holding and switching

3.1.2.91 typedef enum _tUeiOSType tUeiOSType

OS Type

3.1.2.92 typedef enum _tUeiPTPState tUeiPTPState

PTP states for internal PTP state machine

3.1.2.93 typedef enum _tUeiQuadratureDecodingType tUeiQuadratureDecodingType

Method used to count and interpret the pulses the encoder generates on input A and B

3.1.2.94 typedef enum _tUeiQuadratureZeroIndexPhase tUeiQuadratureZeroIndexPhase

The states at which A and B input signals must be in when Z is high to trigger a measurement reset

3.1.2.95 typedef enum _tUeiReferenceResistorType tUeiReferenceResistorType

Reference resistors are used to measure the current flowing through RTDs or other resistive measurement devices. Some terminal block connectors such as the DNA-STP-AIU include a built-in reference resistor. You can also provide your own.

3.1.2.96 typedef enum _tUeiRTDType tUeiRTDType

RTD sensors are specified using the "alpha" (a) constant It is also known as the temperature coefficient of resistance, and symbolizes the resistance change factor per degree of temperature change. The RTD type is used to select the proper coefficients A, B and C for the Callendar Van-Dusen equation used to convert resistance measurements to temperature.

3.1.2.97 typedef enum _tUeiSensorBridgeType tUeiSensorBridgeType

Sensor bridge configurations

3.1.2.98 typedef enum _tUeiSerialPortDataBits tUeiSerialPortDataBits

Serial port number of data bits per character frame

3.1.2.99 typedef enum _tUeiSerialPortFlowControl tUeiSerialPortFlowControl

Serial port flow control setting

3.1.2.100 typedef enum _tUeiSerialPortMinorFrameMode tUeiSerialPortMinorFrameMode

Minor framing configures a delay between groups of characters. The mode selects how minor frames are defined

3.1.2.101 typedef enum _tUeiSerialPortMode tUeiSerialPortMode

Serial port mode

3.1.2.102 typedef enum _tUeiSerialPortParity tUeiSerialPortParity

Serial port parity

3.1.2.103 typedef enum _tUeiSerialPortSpeed tUeiSerialPortSpeed

Serial port pseed

3.1.2.104 typedef enum _tUeiSerialPortStopBits tUeiSerialPortStopBits

Serial port number of stop bits

3.1.2.105 typedef enum _tUeiSerialReadDataType tUeiSerialReadDataType

Data type options for CUeiSerialReader.

3.1.2.106 typedef enum _tUeiSessionState tUeiSessionState

Session state

3.1.2.107 typedef enum _tUeiSessionType tUeiSessionType

Session type

3.1.2.108 typedef enum _tUeiSignalSource tUeiSignalSource

The source of a synchronization signal

3.1.2.109 typedef enum _tUeiStrainGageBridgeType tUeiStrainGageBridgeType

Strain gage bridge configurations

3.1.2.110 typedef enum _tUeiSync1PPSDataType tUeiSync1PPSDataType

Synchronization sessions can read status for different components

3.1.2.111 typedef enum _tUeiSync1PPSMode tUeiSync1PPSMode

Specifies the mode used to obtain 1PPS synchronization clock

3.1.2.112 typedef enum _tUeiSync1PPSOutput tUeiSync1PPSOutput

when synchronizing multiple devices, a master device must output the synchronization 1PPS clock to the slave devices

3.1.2.113 typedef enum _tUeiSync1PPSSource tUeiSync1PPSSource

When in sync clock mode, specifies where the 1PPS clock is coming from

3.1.2.114 typedef enum _tUeiSync1PPSTriggerOperation tUeiSync1PPSTriggerOperation

Trigger operations that a 1PPS sync session can use to start slave devices

3.1.2.115 typedef enum _tUeiSynchroMessageType tUeiSynchroMessageType

Synchro/Resolver message data types (internal use)

3.1.2.116 typedef enum _tUeiSynchroResolverMode tUeiSynchroResolverMode

Specifies whether a synchro or a resolver is measured or simulated

3.1.2.117 typedef enum _tUeiSyncLine tUeiSyncLine

Specifies the backplane synchronization used to connect the hardware to synchronize

3.1.2.118 typedef enum _tUeiTemperatureScale tUeiTemperatureScale

Temperature scales

3.1.2.119 typedef enum _tUeiThermocoupleType tUeiThermocoupleType

Thermocouple types

3.1.2.120 typedef enum _tUeiTimingClockSource tUeiTimingClockSource

Clock source for a timed operation

3.1.2.121 typedef enum _tUeiTimingDuration tUeiTimingDuration

Duration of a timed operation

3.1.2.122 typedef enum _tUeiTimingMode tUeiTimingMode

Mode of a timed operation

3.1.2.123 typedef enum _tUeiTriggerAction tUeiTriggerAction

Trigger Action

3.1.2.124 typedef enum _tUeiTriggerCondition tUeiTriggerCondition

Trigger Condition

3.1.2.125 typedef enum _tUeiTriggerSource tUeiTriggerSource

Trigger Source

3.1.2.126 typedef enum _tUeiVRAPTMode tUeiVRAPTMode

Configures the adaptive peak threshold (APT) mode on variable reluctance sessions. APT finds the point in time where the VR sensor output voltage falls below a certain threshold. This point marks the beginning of the gap between two teeth.

3.1.2.127 typedef enum _tUeiVRDataType tUeiVRDataType

VR sessions can read single measurements, FIFO based data, and AI FIFO data

3.1.2.128 typedef enum _tUeiVRDigitalSource tUeiVRDigitalSource

The source of a digital output on a variable reluctance device

3.1.2.129 typedef enum _tUeiVRFIFOMode tUeiVRFIFOMode

Variable reluctance FIFO data can be used to calculate a map of the inter-tooth delays around the encoder wheel. this is useful to calculate acceleration.

3.1.2.130 typedef enum _tUeiVRMode tUeiVRMode

Configures the input mode on variable reluctance sessions. Variable reluctance input device converts the VR sensor analog signal to a digital signal that can be processed by a counter/timer The VR-608 can use four different modes to measure velocity, position or direction The counting mode can be set to: Decoder: Even and Odd channels are used in pair to determine direction and position Timed: Count number of teeth detected during a timed interval N pulses: Measure the time taken to detect N teeth (Number of teeth needs to be set separately) Z pulse: Measure the number of teeth and the time elapsed between two Z pulses (The Z tooth is usually a gap or a double tooth on the encoder wheel)

3.1.2.131 typedef enum _tUeiVRZCMode tUeiVRZCMode

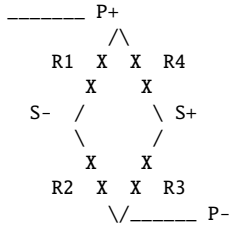
Configures the zero-crossing mode on variable reluctance sessions. Zero crossing finds the point in time where the VR sensor output voltage goes from positive to negative voltage. This point is the center of the tooth lining up with the center of the VR sensor

3.1.2.132 typedef enum _tUeiWatchDogCommand tUeiWatchDogCommand

Configures the watchdog timer to reset the device it is attached to when a programmable timer delay expires. Periodically clearing the timer prevents reset.

3.1.2.133 typedef enum _tUeiWheatstoneBridgeBranch tUeiWheatstoneBridgeBranch

Specify one of the four branches on a Wheatstone bridge



3.1.2.134 typedef enum _tUeiWiringScheme tUeiWiringScheme

Sensor with excitation wiring schemes

3.1.3 Enumeration Type Documentation

3.1.3.1 enum _tUeiAIChannelInputMode

Input modes for Analog input channels

Enumerator:

UeiAIChannelInputModeDifferential Differential mode.
UeiAIChannelInputModeSingleEnded Referenced Single Ended mode.

3.1.3.2 enum _tUeiAIChannelMuxPos

Mux modes for AI self-test

Enumerator:

UeiAIChannelMuxPosOff Normal operations, BIT testing turned off.
UeiAIChannelMuxPosAlt5V Self-test: 5V alternate input to PGA. Input signal has no effect.

UeiAIChannelMuxPosIn5V Self-test: 5V normal input to PGA. Input signal may effect this (useful for detecting broken wiring).

3.1.3.3 enum _tUeiAOAuxCommand

AO sessions use those commands to control auxiliary features such as circuit breakers, short or open circuit simulations

Enumerator:

UeiAOAuxResetCircuitBreaker Re-engage circuit breaker in case it has tripped.

UeiAOAuxShortOpenCircuit Short/Open output lines.

3.1.3.4 enum _tUeiAODACMode

Mode of the DAC connected to channel

Enumerator:

UeiAODACModeDisconnected Disconnects all DACs from the output pin.

UeiAODACModeBothConnected Connect both DACs to output pin.

UeiAODACModeAConnected Connect DAC A to output pin.

UeiAODACModeBConnected Connect DAC B to output pin.

3.1.3.5 enum _tUeiAODiagChannel

The available diagnostic measurements for each AO channel

Enumerator:

UeiAODiagnosticCurrent Monitor output current.

UeiAODiagnosticDACAVoltage Monitor voltage out of DAC A (before relay).

UeiAODiagnosticDACBVoltage Monitor voltage out of DAC B (before relay).

UeiAODiagnosticVoltage Monitor output voltage (after relay).

UeiAODiagnosticTemperature Monitor dual DAC die temperature.

UeiAODiagnosticNone Do not monitor.

3.1.3.6 enum _tUeiAOWaveformClockSource

Clock used to pace main DAC

Enumerator:

UeiAOWaveformClockSourceSYNC2 use SYNC2 line for clock

UeiAOWaveformClockSourceSYNC0 use SYNC0 line for clock

UeiAOWaveformClockSourceALT0 use layer channel zero PLL routed to Channel 0 trigger out to clock all channels

UeiAOWaveformClockSourceTMR clock from internal TMR0 timebase

UeiAOWaveformClockSourceDIO0 use channel DIO0 line for clock

UeiAOWaveformClockSourcePLL clock from PLL on the channel

UeiAOWaveformClockSourceSW DAC is clocked by software (DC offset only).

3.1.3.7 enum _tUeiAOWaveformClockSync

Specifies where a clock signal should be routed to synchronize with other channels and/or layers Route signal to TrgOut to synchronize multiple channels on the same AO-364 Route signal to Sync lines to synchronize multiple AO-364s Only valid for channel 0 on AO-364

Enumerator:

UeiAOWaveformClockRouteNone No sync routing.

UeiAOWaveformClockRouteDIO1ToTrgOut Route DIO1/trigger input pin to Channel0 trigger out (channel 0 only).

UeiAOWaveformClockRouteDIO0ToTrgOut Route DIO0/clock input pin to Channel0 trigger out(channel 0 only).

UeiAOWaveformClockRoutePLLToTrgOut Route PLL clock to Channel0 trigger out (channel 0 only).

UeiAOWaveformClockRoutePLLToSYNC2 Route PLL clock to SYNC2 (channel 0 only).

UeiAOWaveformClockRoutePLLToSYNC0 Route PLL clock to SYNC0 (channel 0 only).

3.1.3.8 enum _tUeiAOWaveformCommandType

AO waveform parameters are changed on the fly by sending commands to the AO session

Enumerator:

UeiAOWaveformCommandShape Specify shape of waveform to be generated.

UeiAOWaveformCommandSweep Specify sweep to apply to waveform being generated.

UeiAOWaveformCommandAWF Load arbitrary waveform data for custom waveform.

3.1.3.9 enum _tUeiAOWaveformMode

Timing mode used to generate waveform.

Enumerator:

UeiAOWaveformModeDDS This mode allows immediate change in the waveform frequency at the expense of a higher THD.

UeiAOWaveformModePLL This mode gives the lowest possible THD but requires 500ms to switch frequency.

3.1.3.10 enum _tUeiAOWaveformOffsetClockSource

Clock used to pace offset DAC

Enumerator:

UeiAOWaveformOffsetClockSourceDIO0 use channel DIO0 line for clock

UeiAOWaveformOffsetClockSourceDIO1 use channel DIO1 line for clock

UeiAOWaveformOffsetClockSourceDAC main DAC clock divided is a source of ODAC

UeiAOWaveformOffsetClockSourcePLL PLL is the source of ODAC (independent of main DAC).

UeiAOWaveformOffsetClockSourceSW ODAC is clocked by software (DC offset only).

3.1.3.11 enum _tUeiAOWaveformOffsetTriggerSource

source used to trigger a new period out of the offset DAC

Enumerator:

UeiAOWaveformOffsetTriggerSourceNone no trigger, layer outputs when clock is available (use with NIS clocking)

UeiAOWaveformOffsetTriggerSourceSYNC3 use SYNC3 line as a trigger

UeiAOWaveformOffsetTriggerSourceSYNC1 use SYNC1 line as a trigger

UeiAOWaveformOffsetTriggerSourceALT0 use channel 0 CH0-TIN line for trigger (needs to be connected to a source)

UeiAOWaveformOffsetTriggerSourceDIO1 use channel DIO1 line for trigger

UeiAOWaveformOffsetTriggerSourceSW use software trigger (simultaneous only within single layer)

3.1.3.12 enum _tUeiAOWaveformSweepControl

Command used to control sweep operation

Enumerator:

UeiAOWaveformSweepUpStart Start sweep from lower value to higher value.

UeiAOWaveformSweepDownStart Start sweep from higher value to lower value.

UeiAOWaveformSweepUpDownStart Start sweep from lower value to higher value to lower value.

UeiAOWaveformSweepDownUpStart Start sweep from higher value to lower value to higher value.

UeiAOWaveformSweepStop Stop sweep.

3.1.3.13 enum _tUeiAOWaveformTriggerSource

source used to trigger a new period out of the main DAC

Enumerator:

UeiAOWaveformTriggerSourceNone no trigger, layer outputs when clock is available

UeiAOWaveformTriggerSourceCH0 ch0 will deliver clock triggered upon CH0_TRIGIN line

UeiAOWaveformTriggerSourceSYNC3 use SYNC3 line as a trigger

UeiAOWaveformTriggerSourceSYNC1 use SYNC1 line as a trigger

UeiAOWaveformTriggerSourceALT0 use channel 0 CH0-TIN line for trigger (nneds to be connected to a source)

UeiAOWaveformTriggerSourceDIO1 use channel DIO1 line for trigger

UeiAOWaveformTriggerSourceSW use software trigger (simultaneous only within single layer)

3.1.3.14 enum _tUeiAOWaveformType

Shape of the waveform to program on a waveform generation capable device

Enumerator:

UeiAOWaveformTypeSine Sine waveform.
UeiAOWaveformTypePulse Square and trapezoid waveform.
UeiAOWaveformTypeTriangle Triangle waveform.
UeiAOWaveformTypeSawtooth Sawtooth and ramp waveform.
UeiAOWaveformTypeCustom Custom waveform (4096 points max.).

3.1.3.15 enum _tUeiAOWaveformXform

Transform applied to each waveform period

Enumerator:

UeiAOWaveformXformNone Do not apply any transfrom.
UeiAOWaveformXformMirror Mirrors each period of the waveform.
UeiAOWaveformXformInvert Inverts waveform.
UeiAOWaveformXformMirrorAndInvert Mirror and Invert waveform.

3.1.3.16 enum _tUeiARINCMessageType

Specifies the type of message sent or received by the ARINC stream

Enumerator:

UeiARINCMessageTypeWord Send/Receive ARINC words.
UeiARINCMessageTypeScheduledWord Modify scheduled ARINC words.
UeiARINCMessageTypePauseScheduler Pause Scheduler.
UeiARINCMessageTypeResumeScheduler Resume Scheduler.
UeiARINCMessageTypeSetMajorMinorActivePage Set the mask.

3.1.3.17 enum _tUeiARINCPortFrameCountingMode

ARINC port frame counting mode

Enumerator:

UeiARINCFrameCountAll Count all frames.
UeiARINCFrameCountGood Only count correcty received frames.
UeiARINCFrameCountFIFO Only count frames placed into the FIFO.
UeiARINCFrameCountTrigger Only count frames that triggered scheduler.
UeiARINCFrameParityError Only count frames with parity error.

3.1.3.18 enum _tUeiARINCPortParity

ARINC port parity

Enumerator:

UeiARINCParityNone No parity bit.

UeiARINCParityOdd Set parity bit to 1 if the frame contains an even number of bit(s) set to 1.

UeiARINCParityEven Set parity bit to 0 if the frame contains an odd number of bit(s) set to 1.

3.1.3.19 enum _tUeiARINCPortSpeed

ARINC port speed

Enumerator:

UeiARINCBitsPerSecond12500 12500 bits per second

UeiARINCBitsPerSecond100000 100000 bits per second

3.1.3.20 enum _tUeiARINCSchedulerType

Specifies the type of scheduler used to emit scheduled words out of a given ARINC-429 port

Enumerator:

UeiARINCSchedulerTypeNormal Normal ARINC scheduler.

UeiARINCSchedulerTypeFramed Framed ARINC scheduler.

UeiARINCSchedulerTypeMajorMinorFrame Major/Minor frame ARINC scheduler.

3.1.3.21 enum _tUeiCANFrameFormat

CAN frame format

Enumerator:

UeiCANFrameBasic CAN 2.0a - basic CAN frame format (11 bit identifier).

UeiCANFrameExtended CAN 2.0b - extended CAN frame format (29 bit identifier).

3.1.3.22 enum _tUeiCANFrameType

Specifies the type of CAN frame sent or received

Enumerator:

UeiCANFrameTypeData Data CAN frame.

UeiCANFrameTypeRemote Remote CAN frame.

UeiCANFrameTypeError Error CAN frame.

3.1.3.23 enum _tUeiCANPortMode

CAN operation mode

Enumerator:

UeiCANPortModeNormal Normal mode.
UeiCANPortModePassive Passive, listen only mode.

3.1.3.24 enum _tUeiCANPortSpeed

CAN port speed

Enumerator:

UeiCANBitsPerSecond10K 10000 bits per second
UeiCANBitsPerSecond20K 20000 bits per second
UeiCANBitsPerSecond50K 50000 bits per second
UeiCANBitsPerSecond100K 100000 bits per second
UeiCANBitsPerSecond125K 125000 bits per second
UeiCANBitsPerSecond250K 250000 bits per second
UeiCANBitsPerSecond500K 500000 bits per second
UeiCANBitsPerSecond800K 800000 bits per second
UeiCANBitsPerSecond1M 1000000 bits per second

3.1.3.25 enum _tUeiColdJunctionCompensationType

Some terminal block connectors such as the DNA-STP-AIU include a built-in cold junction sensor. You can also provide your own.

Enumerator:

UeiCJCTypeBuiltIn Use the CJC sensor built-in the connector block.
UeiCJCTypeConstant Use a constant value for cold-junction temperature.
UeiCJCTypeResource Use a resource string to describe how to measure cold junction temperature.

3.1.3.26 enum _tUeiCounterCaptureTimebase

timebase used measure period, frequency and pulse width

Enumerator:

UeiCounterCaptureTimebase1x Use standard timebase.
UeiCounterCaptureTimebase2x Use double resolution timebase if supported by device.
UeiCounterCaptureTimebaseSync0 Use sync0 as timebase.
UeiCounterCaptureTimebaseSync1 Use sync1 as timebase.
UeiCounterCaptureTimebaseSync2 Use sync2 as timebase.
UeiCounterCaptureTimebaseSync3 Use sync3 as timebase.

3.1.3.27 enum _tUeiCounterGate

source of the gating signal

Enumerator:

UeiCounterGateInternal Use internal gate driven by software.
UeiCounterGateExternal Use external gate connector.

3.1.3.28 enum _tUeiCounterGateMode

gating mode

Enumerator:

UeiCounterGateModeContinuous The counter runs continuously after the gate is set.
UeiCounterGateModeSingleShot The counter runs once after the gate is set.

3.1.3.29 enum _tUeiCounterMode

Counter/Timer mode

Enumerator:

UeiCounterModeCountEvents Event counting.
UeiCounterModeMeasurePulseWidth Pulse width measurement.
UeiCounterModeMeasurePeriod Period measurement.
UeiCounterModeGeneratePulse Pulse generation.
UeiCounterModeGeneratePulseTrain Pulse train generation.
UeiCounterModePulseWidthModulation Pulse width modulation.
UeiCounterModeQuadratureEncoder Quadrature encoder measurement.
UeiCounterModeDirectionCounter Count up or down depending on the state of gate input.

UeiCounterModeTimedPeriodMeasurement Measure period over a specified duration.
UeiCounterModeBinCounting Count events over a specified duration.

3.1.3.30 enum _tUeiCounterSource

Source of the clock signal or the signal to count

Enumerator:

UeiCounterSourceClock Use internal clock as a source.
UeiCounterSourceClockDiv2 Use internal clock divided by 2 as a source.
UeiCounterSourceInput Use external input connector as a source.
UeiCounterSourceCounter0Out Use Counter 0 output as a source.
UeiCounterSourceCounter1Out Use Counter 1 output as a source.
UeiCounterSourceCounter2Out Use Counter 2 output as a source.

3.1.3.31 enum _tUeiCoupling

The type of coupling used to connect a signal to an analog input device

Enumerator:

UeiCouplingDC DC coupling.

UeiCouplingAC AC coupling using 10Hz High pass filter.

UeiCouplingAC_1Hz AC coupling using 1Hz High pass filter.

UeiCouplingAC_100mHz AC coupling using 0.1Hz High pass filter.

3.1.3.32 enum _tUeiCSDBDataType

CSDB sessions can read/write entire frame or single message block

Enumerator:

UeiCSDBDataFrame Read/Write entire frame.

UeiCSDBDataMessageByIndex Read/Write single message selected by aro based index within the frame.

UeiCSDBDataMessageByAddress Read single message selected by address.

3.1.3.33 enum _tUeiCustomScaleType

The type of cutom scale to apply to a scaled input channel

Enumerator:

UeiCustomScaleNone No custom scaling.

UeiCustomScaleLinear Use linear scale.

UeiCustomScalePolynomial Use polynomial scale.

UeiCustomScaleTable Use interpolation table.

3.1.3.34 enum _tUeiDataStreamRelativeTo

Specifies what position to use as a reference in the framework's internal buffer

Enumerator:

UeiDataStreamRelativeToCurrentPosition Read/write data at the specified offset relative to the last sample read or written.

UeiDataStreamRelativeToMostRecentSample Read/write data at the specified offset relative to the most current sample acquired or generated.

3.1.3.35 enum _tUeiDigitalEdge

Edge of a digital pulse

Enumerator:

UeiDigitalEdgeRising Detect rising edge.
UeiDigitalEdgeFalling Detect falling edge.
UeiDigitalEdgeBoth Detect both edges.

3.1.3.36 enum _tUeiDigitalInputMux

Mode of the digital input mux installed on Industrial digital input devices.

Enumerator:

UeiDigitalInputMuxTriState voltage supply disconnected from input line
UeiDigitalInputMuxPullUp voltage supply is connected to input line through a pull-up resistor
UeiDigitalInputMuxDiag voltage supply is directly connected to input line for diagnostic

3.1.3.37 enum _tUeiDigitalTermination

specified termination resistor applied to a digital output

Enumerator:

UeiDigitalTerminationNone No termination.
UeiDigitalTerminationPullUp termination resistor is connected between VCC and digital line
UeiDigitalTerminationPullDown termination resistor is connected between Ground and digital line
UeiDigitalTerminationPullUpPullDown termination resistors are connected between digital line and Ground and VCC

3.1.3.38 enum _tUeiDMMFIRCutoff

Enumerator:

UeiDMMFIROff Disabled.
UeiDMMFIR24Hz 24Hz
UeiDMMFIR50Hz 50Hz,
UeiDMMFIR100Hz 100Hz
UeiDMMFIR1kHz 1kHz

3.1.3.39 enum _tUeiDMMMeasurementMode

the modes of measurement for a DMM device

3.1.3.40 enum _tUeiDMMZeroCrossingMode

Enumerator:

UeiDMMZeroCrossingModeLevel User provided ZC level.
UeiDMMZeroCrossingModeMinMax Use MAX+MIN/2 for ZC level.
UeiDMMZeroCrossingModeRMS Use one period DC average for ZC level.
UeiDMMZeroCrossingModeDC Use DC average for ZC level.

3.1.3.41 enum _tUeiDOPWMMode

Specifies the PWM mode for DO channels that support the capability With PWM mode you can replace the rising and falling edges with a pulse train to allow for soft start and/or stop

Enumerator:

UeiDOPWMDisabled Disable pulsed transitions.
UeiDOPWMSoftStart Use a pulse train for low to high transitions.
UeiDOPWMSoftStop Use a pulse train for high to low transition.
UeiDOPWMSoftBoth Use a pulse train for both transitions.
UeiDOPWMContinuous Continuously output a pulse train.
UeiDOPWMGated Output pulse train only when output line is toggled by software.

3.1.3.42 enum _tUeiDOPWMOutputMode

specifies if PWM is enabled on low size, high size or both sides of the FET This setting is only valid for digital outputlines based on FET

Enumerator:

UeiDOPWMOutputPush PWM enabled on high side only.
UeiDOPWMOutputPull PWM enabled on low side only.
UeiDOPWMOutputPushPull PWM enabled on both sides.
UeiDOPWMOutputOff PWM output is off.

3.1.3.43 enum _tUeiEEPROMArea

Specifies the area to read/write in EEPROM

Enumerator:

UeiEEPROMAreaCommon Common area (holds data common to all devices).
UeiEEPROMAreaCalibration Calibration coefficients area.
UeiEEPROMAreaInitialization Initialization state area.
UeiEEPROMAreaOperation Operation mode state area.
UeiEEPROMAreaShutdown Shutdown state area.
UeiEEPROMAreaNames Channel names area.
UeiEEPROMAreaWhole Read whole EEPROM if layer supports it.
UeiEEPROMAreaFlags Flags area.

3.1.3.44 enum _tUeiEvent

Asynchronous event

Enumerator:

UeiEventFrameDone A frame has been filled.

UeiEventBufferDone All the frames of a buffer have been filled.

UeiEventError An error occurred while the session was running.

UeiEventDigitalIn Digital input line state change detected.

UeiEventSessionDone The session is done, it either acquired or generated the specified amount of data.

UeiEventFIFOWatermark Watermark level is reached in the FIFO.

UeiEventPeriodicTimer Periodic timer expired.

3.1.3.45 enum _tUeiFeatureEnable

Manual mode lets you turn on or off a feature from your code Automatic mode controls the feature from a setting saved in EEPROM

Enumerator:

UeiFeatureDisabled Feature is disabled.

UeiFeatureEnabled Feature is enabled.

UeiFeatureAuto Feature is enabled/disabled depending on EEPROM content.

3.1.3.46 enum _tUeiFlushAction

Specifies how to flush a messaging port (serial, CAN, ARINC, MIL-1553)

Enumerator:

UeiFlushReadBuffer Flush the receive device by writing all data from hardware FIFO to receive buffer.

UeiFlushDiscardReadBuffer Discard the hardware FIFO and receive buffer contents (data is lost).

UeiFlushWriteBuffer Flush the transmit buffer by writing all buffered data to device.

UeiFlushDiscardWriteBuffer Discard the transmit buffer content (data is not sent).

3.1.3.47 enum _tUeiHDLCPortAbortSymbol

Symbol used to abort HDLC transmission

Enumerator:

UeiHDLCPortAbort7 Send 0x7F to abort.

UeiHDLCPortAbort15 Send 0x7FFF to abort.

3.1.3.48 enum _tUeiHDLCPortClockSource

clock source used to synchronize transmit and or receive lines

Enumerator:

UeiHDLCPortClockExternalPin Take clock from RxC/TxC pin (default).

UeiHDLCPortClockBRG Take clock from baud rate generator.

UeiHDLCPortClockDPLL Take clock from DPLL divided by 32.

UeiHDLCPortClockDPLLDiv8 Take clock from DPLL divided by 8.

UeiHDLCPortClockDPLLDiv16 Take clock from DPLL divided by 16.

UeiHDLCPortClockDPLLSRCBRG Take clock from DPLL with DPLL taking its input from baud rate generator.

3.1.3.49 enum _tUeiHDLCPortCRCMode

Algorithm used to calculate the CRC

Enumerator:

UeiHDLCPortCRCNone CRC is not check neither for transmit nor receive.

UeiHDLCPortCRCUser User is responsible for inserting and checking CRC.

UeiHDLCPortCRC16CCITT 16-bit CCITT CRC is used (x^16+x^12+x^5+1)

UeiHDLCPortCRC16 16-bit polynomial CRC is used (x^16+x^15+x^2+1)

UeiHDLCPortCRC32 32-bit Eth CRC is used (x^32+x^26+x^23+x^22+x^16+x^12+x^11+x^10+x^8+x^7+x^5+x

3.1.3.50 enum _tUeiHDLCPortEncoding

HDLC port encoding

Enumerator:

UeiHDLCPortEncodingNRZ NRZ encoding.

UeiHDLCPortEncodingNRZB inverted NRZ encoding

UeiHDLCPortEncodingNRZI NRZI encoding.

UeiHDLCPortEncodingNRZIMark NRZI encoding, invert state for 1.

UeiHDLCPortEncodingNRZISpace NRZI encoding, invert state for 0.

UeiHDLCPortEncodingBiphaseMark biphase encoding, used with DPLL

UeiHDLCPortEncodingBiphaseSpace biphase encoding, used with DPLL

UeiHDLCPortEncodingBiphaseLevel biphase encoding, used with DPLL

UeiHDLCPortEncodingBiphaseDiff biphase encoding, used with DPLL

3.1.3.51 enum _tUeiHDLCPortFilterMode

filter mode

Enumerator:

UeiHDLCPortFilterNone No filtering.

UeiHDLCPortFilterA16 +16 bits into RxFIFO if Addr matches or B/C as 2 bytes

UeiHDLCPortFilterA24 +24 bits into RxFIFO if Addr matches or B/C as 3 bytes

UeiHDLCPortFilterA32 +32 bits into RxFIFO if Addr matches or B/C as 4 bytes

UeiHDLCPortFilterEALS Places bytes while LS==0, then byte with LS==1 then 16 bits as 2 bytes into RxFIFO if EA matches or B/C.

UeiHDLCPortFilterEA24 Places 24 bits as 3 bytes into RxFIFO if EA matches or B/C.

UeiHDLCPortFilterEAMS Places bytes while MS==0, then byte with MS==1 then 8 bits as 1 byte into RxFIFO if Ext Addr matches or B/C.

UeiHDLCPortFilterEAMS16 Places bytes while MS==0, then byte with MS==1 then 16 bits as 2 bytes into RxFIFO if Ext Addr matches or B/C.

3.1.3.52 enum _tUeiHDLCPortIdleCharacter

HDLC port idle character

Enumerator:

UeiHDLCPortIdleFlag continuous flags

UeiHDLCPortIdleZero continuous zeroes

UeiHDLCPortIdleOne continuous ones

UeiHDLCPortIdleMark idle chars are marks

UeiHDLCPortIdleSpace idle chars are spaces

UeiHDLCPortIdleMS alternating Mark and Space

UeiHDLCPortIdle01 alternating 0 and 1

3.1.3.53 enum _tUeiHDLCPortPhysical

HDLC port physical interface

Enumerator:

UeiHDLCPortRS232 RS-232.

UeiHDLCPortRS485 RS-485.

UeiHDLCPortRS422 RS-422.

UeiHDLCPortV35 V35.

3.1.3.54 enum _tUeiHDLCPortPreamble

HDLC port preamble pattern

Enumerator:

UeiHDLCPortPreambleNone No preamble.
UeiHDLCPortPreambleZero All zeros.
UeiHDLCPortPreambleOne All ones.
UeiHDLCPortPreambleFlag All flags.
UeiHDLCPortPreamble10 Alternating 1 and 0.
UeiHDLCPortPreamble01 Alternating 0 and 1.

3.1.3.55 enum _tUeiHDLCPortPreambleSize

HDLC port preamble size

Enumerator:

UeiHDLCPortPreambleSize16 16 bits preamble is used
UeiHDLCPortPreambleSize32 32 bits preamble is used
UeiHDLCPortPreambleSize64 64 bits preamble is used

3.1.3.56 enum _tUeiHDLCPortUnderrunAction

Action to take when underrun condition is detected

Enumerator:

UeiHDLCPortUnderrunFinish Close the frame by adding CRC to it.
UeiHDLCPortUnderrunFlags Send flags.

3.1.3.57 enum _tUeiI2CBusCode

Bus condition codes

Enumerator:

UeiI2CBusUnknown Unknown bus code.
UeiI2CBusStart Start condition, no data.
UeiI2CBusRestart Restart condition, no data.
UeiI2CBusStop Stop, no data.
UeiI2CBusAddressAck Address received with ACK.
UeiI2CBusAddressNack Address received with no ACK.
UeiI2CBusDataAck Data received with ACK.
UeiI2CBusDataNack Data received with no ACK.
UeiI2CBusAllData Last data byte received.
UeiI2CBusClockStretchError Clock stretching error.

3.1.3.58 enum _tUeiI2CCommand

I2C commands

Enumerator:

UeiI2CCommandWrite Write command.

UeiI2CCommandRead Read command.

UeiI2CCommandWriteRead Write, Restart, Read command.

3.1.3.59 enum _tUeiI2CLoopback

I2C loopback modes

Enumerator:

UeiI2CLoopbackNone No loopback enabled.

UeiI2CLoopbackRelay Connect master and slave on channel using relay. Master and slave are still on external I2C bus.

UeiI2CLoopbackFPGA Enable loopback within FPGA. Disconnects master and slave from the external I2C bus.

3.1.3.60 enum _tUeiI2CMessageType

I2C message data types

Enumerator:

UeiI2CMessageSlaveWrite Slave Tx write.

UeiI2CMessageMasterWrite Master Tx write.

UeiI2CMessageSlaveRead Read from Slave Rx.

UeiI2CMessageMasterRead Read from Master Rx.

UeiI2CMessageMasterAvailInput Read number of available master input messages.

UeiI2CMessageSlaveAvailInput Read number of available slave input messages.

UeiI2CMessageSlaveAvailOutput Read number of available slave output messages.

3.1.3.61 enum _tUeiI2CPortSpeed

Enumerator:

UeiI2CBitsPerSecond100K 100 Kbits per second

UeiI2CBitsPerSecond400K 400 Kbits per second

UeiI2CBitsPerSecond1M 1 Mbits per second

UeiI2CBitsPerSecondCustom Use custom.

3.1.3.62 enum _tUeiI2CSlaveAddressWidth

Enumerator:

UeiI2CSlaveAddress7bit 7-bit slave address

UeiI2CSlaveAddress10bit 10-bit slave address

3.1.3.63 enum _tUeiI2CSlaveDataMode

I2C slave data modes

Enumerator:

UeiI2CSlaveDataModeFIFO Slave transmits data using FIFO.

UeiI2CSlaveDataModeRegister Slave transmits up to 4 bytes of pre-configured data.

3.1.3.64 enum _tUeiI2CTTLLevel

Enumerator:

UeiI2CTTLLevel3_3V 3.3V

UeiI2CTTLLevel5V 5V

3.1.3.65 enum _tUeiInitParameter

Specifies the parameter to read/write

Enumerator:

UeiInitParameterModelId IOM model ID.

UeiInitParameterModelOption IOM model option.

UeiInitParameterIOMSerial IOM serial number.

UeiInitParameterIOMMfgDate IOM manufacturing date.

UeiInitParameterIOMBaseFrequency IOM base frequency.

UeiInitParameterTickSize OS tick size.

UeiInitParameterPeriod Periodic tick size.

UeiInitParameterWatchdogTimer Watchdog timer reset delay.

UeiInitParameterTime Current time, seconds from 1/1/2000.

UeiInitParameterOptions Firmware options.

UeiInitParameterComDelay Delay before entering shutdown mode after communication is lost.

UeiInitParameterFWCT Type of the IOM firmware.

UeiInitParameterLayerId Layer ID.

UeiInitParameterLayerOption Layer option.

UeiInitParameterLayerSerial Layer serial number.

UeiInitParameterEEPROMSize Total EEPROM size.

UeiInitParameterLayerMfgDate Layer manufacturing date.

UeiInitParameterLayerCalDate Layer calibration date.

UeiInitParameterLayerCalExpiration Layer calibration expiration date.

UeiInitParameterDQRev DaqBIOS revision supported.

UeiInitParameterFWRev Firmware revision supported.

3.1.3.66 enum _tUeiIRIGDecoderInputType

The type of signal connected to time decoder input

Enumerator:

UeiIRIGDecoderInputAM Time code is provided as an AM signal.

UeiIRIGDecoderInputManchesterRF0 Time code is provided as a Manchester II code on RF input 0.

UeiIRIGDecoderInputManchesterRF1 Time code is provided as a Manchester II code on RF input 1.

UeiIRIGDecoderInputManchesterIO0 Time code is provided as a Manchester II code on I/O input 0.

UeiIRIGDecoderInputManchesterIO1 Time code is provided as a Manchester II code on I/O input 1.

UeiIRIGDecoderInputNRZRF0 Time code is provided as a NRZ code on RF input 0.

UeiIRIGDecoderInputNRZRF1 Time code is provided as a NRZ code on RF input 1.

UeiIRIGDecoderInputNRZIO0 Time code is provided as a NRZ code on I/O input 0.

UeiIRIGDecoderInputNRZIO1 Time code is provided as a NRZ code on I/O input 1.

UeiIRIGDecoderInputGPS Time code is taken from GPS NMEA message.

3.1.3.67 enum _tUeiIRIGDOTTLSource

Specifies the source that drives an IRIG TTL output

Enumerator:

UeiIRIGDOTTLAMtoNRZ AM->NRZ output.

UeiIRIGDOTTLGPSFixValid GPS Fix Valid output.

UeiIRIGDOTTLGPSAntennaShorted GPS Antenna Shorted output.

UeiIRIGDOTTLGPSAntennaOK GPS Antenna Ok output.

UeiIRIGDOTTLGPSTxD1 GPS TXD1 (COM1) output.

UeiIRIGDOTTLGPSTxD0 GPS TXD0 (COM0) output.

UeiIRIGDOTTLManchesterIItoNRZ Decoded Manchester II -> NRZ sequence.

UeiIRIGDOTTLSYNC3 Drive output from sync[3].

UeiIRIGDOTTLSYNC2 Drive output from sync[2].

UeiIRIGDOTTLSYNC1 Drive output from sync[1].

UeiIRIGDOTTLSYNC0 Drive output from sync[0].

UeiIRIGDOTTLOutputCarrierFrequency Output carrier frequency.

UeiIRIGDOTTLPLLFrequency Custom frequency output (from PLL).

UeiIRIGDOTTLPrecision10MHZ Precision 10MHz.

UeiIRIGDOTTLPrecision5MHZ Precision 5MHz.

UeiIRIGDOTTLPrecision1MHZ Precision 1MHz.

UeiIRIGDOTTLNRZStartStrobe Output NRZ start strobe.

UeiIRIGDOTTLManchesterIITimeCode Manchester II output time code.

UeiIRIGDOTTLNRZTimeCode NRZ output time code.

UeiIRIGDOTTLGPS1PPS Re-route GPS 1PPS pulse.

UeiIRIGDOTTL1PPH 1PPH pulse

UeiIRIGDOTTL1PPM 1PPM pulse

UeiIRIGDOTTL1PPS 1PPS pulse

UeiIRIGDOTTL0_1S 0.1sec pulse

UeiIRIGDOTTL0_01S 0.01sec pulse

UeiIRIGDOTTL1uS 1uec pulse

UeiIRIGDOTTLLogic1 Drive output with 1.

UeiIRIGDOTTLLogic0 Drive output with 0.

UeiIRIGDOTTLEventChannel1 Drive output from event channel 0.

UeiIRIGDOTTLEventChannel2 Drive output from event channel 1.

UeiIRIGDOTTLEventChannel3 Drive output from event channel 3.

3.1.3.68 enum _tUeiIRIGEventSource

Specifies the source that drives the event module

Enumerator:

UeiDaqIRIGEventSYNC1 Drive event module from SYNC bus line 0.

UeiDaqIRIGEventSYNC2 Drive event module from SYNC bus line 1.

UeiDaqIRIGEventSYNC3 Drive event module from SYNC bus line 2.

UeiDaqIRIGEvent1PPS Drive event module from SYNC bus line 3.

UeiDaqIRIGEventEXTT Drive event module from PPS pulse.

UeiDaqIRIGEventEINV Drive event module from External time received and applied.

UeiDaqIRIGEventTTL0 Drive event module from External sync lost.

UeiDaqIRIGEventTTL1 Drive event module from External TTL input 0.

UeiDaqIRIGEventTTL2 Drive event module from External TTL input 1.

UeiDaqIRIGEventTTL3 Drive event module from External TTL input 3.

3.1.3.69 enum _tUeiIRIGTimeCodeFormat

The time code format used by the time coder and the time decoder

Enumerator:

UeiIRIGTimeCodeFormatA IRIG-A.
UeiIRIGTimeCodeFormatB IRIG-B.
UeiIRIGTimeCodeFormatD_100Hz IRIG-D 100Hz.
UeiIRIGTimeCodeFormatD_1000Hz IRIG-D 1000Hz.
UeiIRIGTimeCodeFormatE_100Hz IRIG-E 100Hz.
UeiIRIGTimeCodeFormatE_1000Hz IRIG-E 1000Hz.
UeiIRIGTimeCodeFormatG IRIG-G.
UeiIRIGTimeCodeFormatH_100Hz IRIG-H 100Hz.
UeiIRIGTimeCodeFormatH_1000Hz IRIG-H 1000Hz.

3.1.3.70 enum _tUeiIRIGTimeFormat

Specifies the format used to send/receive time to/from the IRIG device

Enumerator:

UeiIRIGBCDTime Time is formatted as binary coded decimal values.
UeiIRIGSBSTime Time is formatted as straight binary seconds values.
UeiIRIGANSITime Time is formatted as struct tm C ANSI.
UeiIRIGGPSTime Time (and position) is formatted as a NMEA GPS string.
UeiIRIGTREG Content of TREG registers (internal use only).
UeiIRIGTimeCode Raw timecode data stream.
UeiIRIGSecondsSinceUnixEpochTime Time is formated as number of seconds since UNIX EPOCH.

3.1.3.71 enum _tUeiIRIGTimeKeeper1PPSSource

The source of the incoming 1 PPS signal used to keep time

Enumerator:

UeiIRIG1PPSInternal 1PPS signal is generated internally with precision oscillator
UeiIRIG1PPSInputTimeCode 1PPS signal is derived from input timecode
UeiIRIG1PPSGPS 1PPS signal is derived from GPS
UeiIRIG1PPSRFIn 1PPS signal is derived from signal connected on RF input
UeiIRIG1PPSExternalTTL0 External 1PPS signal is connected to TTL0 input pin.
UeiIRIG1PPSExternalTTL1 External 1PPS signal is connected to TTL1 input pin.
UeiIRIG1PPSExternalTTL2 External 1PPS signal is connected to TTL2 input pin.
UeiIRIG1PPSExternalTTL3 External 1PPS signal is connected to TTL3 input pin.
UeiIRIG1PPSExternalSync0 External 1PPS signal is connected to SYNC0 bus line.
UeiIRIG1PPSExternalSync1 External 1PPS signal is connected to SYNC1 bus line.
UeiIRIG1PPSExternalSync2 External 1PPS signal is connected to SYNC2 bus line.
UeiIRIG1PPSExternalSync3 External 1PPS signal is connected to SYNC3 bus line.

3.1.3.72 enum _tUeiLogFileFormat

The format for logging acquired data to disk

Enumerator:

UeiLogFileCSV Comma separated values format.

UeiLogFileBinary Binary format.

3.1.3.73 enum _tUeiLVDTWiringScheme

LVDT/RVDT excitation wiring schemes

Enumerator:

UeiLVDTFourWires Sensor is connected using four wires.

UeiLVDTFiveWires Sensor is connected using five wires.

3.1.3.74 enum _tUeiMeasurementType

Measurement type

Enumerator:

UeiMeasurementVoltage Voltage measurement.

UeiMeasurementThermocouple Temperature measurement with thermocouples.

UeiMeasurementStrainGage Strain gage measurement.

UeiMeasurementVoltageWithExcitation Voltage measurement with excitation (typically for load cells).

UeiMeasurementRTD Temperature measurement with RTDs.

UeiMeasurementResistance Resistance measurement.

UeiMeasurementAccelerometer Accelrometer measurement.

UeiMeasurementLVDT Position Measurement with LVDT or RVDT.

UeiMeasurementSynchroResolver Angle measurement with Synchro or Resolver.

UeiMeasurementCurrent Current measurement.

3.1.3.75 enum _tUeiMeasurementUnits

the units of measurement for a device

3.1.3.76 enum _tUeiMIL1553A708Ops

ARINC-708 operations

Enumerator:

UeiMIL1553A708OpDisable Disable ARINC-708 operation.

UeiMIL1553A708OpEnable Enable ARINC-708 operation.

UeiMIL1553A708FIFODisable Disable ARINC-708 FIFO mode operation.

UeiMIL1553A708FIFOEnable Enable ARINC-708 FIFO mode operation.

UeiMIL1553A708FIFOClear Clear ARINC-708 FIFOs.

3.1.3.77 enum _tUeiMIL1553BCFrameType

MIL-1553 BC frame type

Enumerator:

UeiMIL1553BCFrameUndef Undefined frame.

UeiMIL1553BCFrameMajor Major frame.

UeiMIL1553BCFrameMinor Minor frame.

3.1.3.78 enum _tUeiMIL1553BCGotoOps

1553 BC GoTo command types

Enumerator:

UeiMIL1553GotoBcbNoret Make jump after BCB, no return.

UeiMIL1553GotoMnNoret Make jump after minor frame, no return.

UeiMIL1553GotoBcbBcb Make jump after BCB, return after BCB.

UeiMIL1553GotoMnBCB Make jump after minor frame, return after BCB.

UeiMIL1553GotoBcbMn Make jump after BCB, return after minor frame.

UeiMIL1553GotoMnMn Make jump after minor frame, return after minor frame.

3.1.3.79 enum _tUeiMIL1553BCMajorFlags

MIL-1553 BC major frame flags

Enumerator:

UeiMIL1553MjLink Do not wait for the MN clock to execute this entry.

UeiMIL1553MjEnable Major entry is enabled.

UeiMIL1553MjExecuteOnce Execute entry once, then disable it.

UeiMIL1553MjSendMessage Send asynchronous message to user when executed.

UeiMIL1553MjSwapEnabled Swap this entry on the next swap cycle (enable if disabled and vice versa).

UeiMIL1553MjDoneOnce (status) Entry was executed at least once (R/O)

3.1.3.80 enum _tUeiMIL1553BCMinorFlags

MIL-1553 BC minor frame flags

Enumerator:

UeiMIL1553MnEnable Minor entry is enabled.

UeiMIL1553MnExecuteOnce Execute entry once, then disable it.

UeiMIL1553MnCurrentBusB (status) Bus B selected as a current bus

UeiMIL1553MnRTRecovered (status) RT had failed but since recovered

UeiMIL1553MnNDoneOnce (status) Entry was executed at least once

UeiMIL1553MnExecError (status) Label execution error

3.1.3.81 enum _tUeiMIL1553BCOps

1553 BC operations

Enumerator:

UeiMIL1553BcOpDisable Disable BC operation.

UeiMIL1553BcOpEnable Enable BC operation.

UeiMIL1553BcOpStepMj Execute one step (major).

UeiMIL1553BcOpStepMn Execute one step (minor).

UeiMIL1553BcOpGoto Switch to a different entry.

UeiMIL1553BcOpMnSelect Select minor frame blocks.

UeiMIL1553BcOpMjSwap Swap major frame "swap" marked entries from "disabled" to "enabled".

3.1.3.82 enum _tUeiMIL1553BCRetryType

STD-MIL-1553 commands

Enumerator:

UeiMIL1553BCR_IRT Retry on incorrect RT# in status.

UeiMIL1553BCR_RUS Retry on unexpected status reception.

UeiMIL1553BCR_RUD Retry on unexpected data reception.

UeiMIL1553BCR_RWB Retry on wrong bus response.

UeiMIL1553BCR_RIS Retry on illegal bits set in status.

UeiMIL1553BCR_RBB Retry on busy bit in status.

UeiMIL1553BCR_RTE Retry on bus timing error.

UeiMIL1553BCR_RWC Retry on word count.

UeiMIL1553BCR_RE Reenable command transmission on succesfull status reply.

UeiMIL1553BCR_RNR Retry on no-response.

UeiMIL1553BCR_ERE Enable re-transmit on alternative bus each retry.

UeiMIL1553BCR_ESR Enable periodic status request command when retry.

3.1.3.83 enum _tUeiMIL1553CommandType

STD-MIL-1553 commands

Enumerator:

UeiMIL1553CmdBCRT Remote terminal to receive data from bus controller.

UeiMIL1553CmdRTBC Remote terminal to transmit data to bus controller.

UeiMIL1553CmdRTRT One remote terminal to transmit data to another remote terminal.

UeiMIL1553CmdModeTxNoData Status word.

UeiMIL1553CmdModeTxWithData Remote terminal to transmit data and/or status word to bus controller.

UeiMIL1553CmdModeRxWithData Remote terminal to receive data for mode command from bus controller.

UeiMIL1553CmdBCRTBroadcast Remote terminal to receive broadcast data from bus controller.

UeiMIL1553CmdRTRTBroadcast One remote terminal to broadcast data to other remote terminal.

UeiMIL1553CmdModeTxNoDataBroadcast Mode command without data, remote terminals should not reply.

UeiMIL1553CmdModeRxWithDataBroadcast Mode command with data, remote terminals should receive data.

3.1.3.84 enum _tUeiMIL1553Endian

1553 port endian

Enumerator:

UeiMIL1553SmallEndian Use small-endian notation for Rx or Tx (ARINC-708 only).

UeiMIL1553BigEndian Use big-endian notation for Rx or Tx.

3.1.3.85 enum _tUeiMIL1553FilterType

1553 filter settings

Enumerator:

UeiMIL1553FilterByRt Filter only by RT numbers, all SAs of any lenght are enabled.

UeiMIL1553FilterByRtSa Each RT/SA combination should be declared.

UeiMIL1553FilterByRtSaSize Each RT/SA combination and size of Rx/Tx Min/Max data should be declared.

UeiMIL1553FilterValidationEntry Validation entry is written directly.

3.1.3.86 enum _tUeiMIL1553FrameType

Specifies the type of 1553 frame sent or received

Enumerator:

UeiMIL1553FrameTypeRtData RW Remote terminal data (tUeiMIL1553RTFrame).

UeiMIL1553FrameTypeBusWriter W Data to transmit to output FIFO (tUeiMIL1553BusWriterFrame).

UeiMIL1553FrameTypeBusMon R Data in bus monitor format (tUeiMIL1553BMFrame).

UeiMIL1553FrameTypeBusMonCmd R Data in protocol-parsed bus monitor format (tUeiMIL1553BMCmdFrame).

UeiMIL1553FrameTypeRtStatusData R Remote terminal status data (ready/sent) (tUeiMIL1553RTStatusFrame).

UeiMIL1553FrameTypeRtStatusLast R Bus status and last command and last mode command - reserved.

UeiMIL1553FrameTypeRtParameters W Remote terminal parameters data (tUeiMIL1553RTParametersFrame).

UeiMIL1553FrameTypeRtControlData W Remote terminal control data (tUeiMIL1553RTControlFrame).

UeiMIL1553FrameTypeError R Error 1553 frame - UEIDAQ_NOT_IMPLEMENTED_ERROR.

UeiMIL1553FrameTypeBCCBData W BCCB Data (upper 0x80) (tUeiMIL1553BCCBDataFrame).

UeiMIL1553FrameTypeBCCBStatus R BCCB Status (lower 0x40) (tUeiMIL1553BCCBStatusFrame).

UeiMIL1553BCSchedFrame RW Scheduler entries (tUeiMIL1553BCSchedFrame).

UeiMIL1553BCControlFrame W Control BC (tUeiMIL1553BCControlFrame).

UeiMIL1553BCStatusFrame R BC Status (tUeiMIL1553BCStatusFrame).

UeiMIL1553FrameTypeA708Data RW ARINC-708 Data (tUeiMIL1553A708DataFrame).

UeiMIL1553FrameTypeA708Control W ARINC-708 Control (tUeiMIL1553A708ControlFrame).

UeiMIL1553FrameTypeGeneric RW Generic frame type (tUeiMIL1553Frame).

3.1.3.87 enum _tUeiMIL1553ModeCommandTypes

STD-MIL-1553 mode command codes. These mode codes are encoded in the Word Count field

Enumerator:

UeiMIL1553ModeDynamicBusControl DBC, Data=No, Broadcast=No.

UeiMIL1553ModeSynchronize Synchronize terminals, Data=No, Broadcast=Yes.

UeiMIL1553ModeTransmitStatusWord Transmit status word for RT, Data=No, Broadcast=No.

UeiMIL1553ModeStartSelfTest Initiate self test, Data=No, Broadcast=Yes.

UeiMIL1553ModeTxShutdown Transmitter shutdown, Data=No, Broadcast=Yes.

UeiMIL1553ModeTxShutdownOver Transmitter shutdown override, Data=No, Broadcast=Yes.

UeiMIL1553ModeInhTerminalFlag Inhibit terminal flag bit, Data=No, Broadcast=Yes.

UeiMIL1553ModeInhTerminalFlagOver Override Inhibit terminal flag bit, Data=No, Broadcast=Yes.

UeiMIL1553ModeResetRt Reset remote terminal, Data=No, Broadcast=Yes.

UeiMIL1553ModeTxVectorWord Transmit vector word, Data=Yes, Broadcast=No.

UeiMIL1553ModeSynchronizeData Synchronize with synchronization data, Data=Yes, Broadcast=Yes.

UeiMIL1553ModeTxLastCommand Transmit last command word, Data=Yes, Broadcast=No.

UeiMIL1553ModeTxBITWord Transmit vector word, Data=Yes, Broadcast=No.

UeiMIL1553ModeSelectedTxShutdown Shutdown selected transmitter, Data=Yes, Broadcast=Yes.

UeiMIL1553ModeOverrideTxShutdown Override selected transmitters shutdown, Data=Yes, Broadcast=Yes.

3.1.3.88 enum _tUeiMIL1553PortActiveBus

1553 port active bus

Enumerator:

UeiMIL1553OpModeBusA Bus A is used (default for transmission).

UeiMIL1553OpModeBusB Bus B is used.

UeiMIL1553OpModeBusBoth Bus Controller Mode (default for listening).

3.1.3.89 enum _tUeiMIL1553PortCoupling

1553 port coupling

Enumerator:

UeiMIL1553CouplingDisconnected Device is disconnected from the bus.

UeiMIL1553CouplingTransformer Bus is connected using transformer.

UeiMIL1553CouplingLocalStub Stub is located right on the PCB (requres special hardware).

UeiMIL1553CouplingDirect Direct coupling - no transformer.

3.1.3.90 enum _tUeiMIL1553PortOpMode

1553 port operating mode

Enumerator:

UeiMIL1553OpModeBusMonitor Bus Monitor Mode.

UeiMIL1553OpModeRemoteTerminal Remote Terminal Mode.

UeiMIL1553OpModeBusController Bus Controller Mode.

UeiMIL1553OpModeARINC708 ARINC-708 Mode (1553 is shut off).

3.1.3.91 enum _tUeiMIL1553RTControlType

STD-MIL-1553 control commands

Enumerator:

UeiMIL1553Disable Disable RT functionality.

UeiMIL1553Enable Enable RT functionality.

UeiMIL1553EnableMask Enable RT functionality by mask.

UeiMIL1553RTSetRtBcBlock Select which block to use for data for RT->BC command (Tx).

UeiMIL1553RTSetBcRtBlock Select which block to use for data for BC->RT command (Rx).

UeiMIL1553RTSetValid Write validation entry directly to RT.

UeiMIL1553RTResponseTiming Set inter message gap.

3.1.3.92 enum _tUeiMuxDmmMode

MUX output to DMM modes

3.1.3.93 enum _tUeiMUXMessageType

MUX message type

3.1.3.94 enum _tUeiMuxSyncOutputMode

MUX devices can assert a synchronization output line when relays are configured

Enumerator:

UeiMuxSyncOutputLogic0 Drive constant logic 0.

UeiMuxSyncOutputLogic1 Drive constant logic 1.

UeiMuxSyncOutputSyncLine0 Driven by internal sync bus line #0.

UeiMuxSyncOutputSyncLine1 Driven by internal sync bus line #1.

UeiMuxSyncOutputSyncLine2 Driven by internal sync bus line #2.

UeiMuxSyncOutputSyncLine3 Driven by internal sync bus line #3.

UeiMuxSyncOutputRelaysReadyPulse0 Drive logic 0 pulse upon "relays ready" event.

UeiMuxSyncOutputRelaysReadyPulse1 Drive logic 1 pulse upon "relays ready" event.

UeiMuxSyncOutputRelaysReadyLogic0 Driven to logic 0 upon "relays ready" event.

UeiMuxSyncOutputRelaysReadyLogic1 Driven to logic 1 upon "relays ready" event.

3.1.3.95 enum _tUeiMuxVoltage

Voltage levels for DC/DC during relay holding and switching

Enumerator:

UeiMuxVoltage2_42 2.42V

UeiMuxVoltage3_3 3.3V

UeiMuxVoltage4_2 4.2V

UeiMuxVoltage5_1 5.1V

3.1.3.96 enum _tUeiOSType

OS Type

Enumerator:

UeiOSWindows Microsoft Windows NT/2000/XP/2003 operating systems.

UeiOSLinux Linux operating system.

UeiOSPharlapEts Ardence Pharlap ETS operating system.

UeiOSMacOS Apple MacOS operating system.

3.1.3.97 enum _tUeiPTPState

PTP states for internal PTP state machine

3.1.3.98 enum _tUeiQuadratureDecodingType

Method used to count and interpret the pulses the encoder generates on input A and B

Enumerator:

UeiQuadratureDecodingType1x Count one pulse for each rising edge of Input A signal.

UeiQuadratureDecodingType2x Count one pulse for each rising and falling edge of Input A signal.

UeiQuadratureDecodingType4x Count one pulse for each rising and falling edge of both A and B Input signals.

3.1.3.99 enum _tUeiQuadratureZeroIndexPhase

The states at which A and B input signals must be in when Z is high to trigger a measurement reset

Enumerator:

UeiQuadratureZeroIndexPhaseZHigh reset measurement when Z is high

UeiQuadratureZeroIndexPhaseALowBLow reset measurement when Z is high, A is low and B is low

UeiQuadratureZeroIndexPhaseALowBHigh reset measurement when Z is high, A is low and B is high

UeiQuadratureZeroIndexPhaseAHighBLow reset measurement when Z is high, A is high and B is low

UeiQuadratureZeroIndexPhaseAHighBHigh reset measurement when Z is high, A is high and B is high

3.1.3.100 enum _tUeiReferenceResistorType

Reference resistors are used to measure the current flowing through RTDs or other resistive measurement devices. Some terminal block connectors such as the DNA-STP-AIU include a built-in reference resistor. You can also provide your own.

Enumerator:

UeiRefResistorBuiltIn Use the reference resistor built-in the terminal block.

UeiRefResistorExternal Use an external reference resistor connected on a dedicated channel.

3.1.3.101 enum _tUeiRTDType

RTD sensors are specified using the "alpha" (a) constant It is also known as the temperature coefficient of resistance, and symbolizes the resistance change factor per degree of temperature change. The RTD type is used to select the proper coefficients A, B and C for the Callendar Van-Dusen equation used to convert resistance measurements to temperature.

Enumerator:

UeiRTDType3750 Low-cost Platinum RTD. a=0.00375 A=3.81EŰ3 B=-6.02E-7 C=-6.0E-12.

UeiRTDType3850 IEC-751 European standard Platinum RTD. a=0.00385 A=3.9083E-3 B=-5.775E-7 C=-4.183E-12.

UeiRTDType3902 US Industrial standard Platinum RTD. a=0.003902 A=3.96E-3 B=-5.93E-7 C=-4.3E-12.

UeiRTDType3911 ASTM 1137 American standard Platinum RTD. a=0.003911 A=3.9692E-3 B=Ű5.8495E-7 C=Ű4.233E-12.

UeiRTDType3916 JISC-1604 Japanese standard Platinum RTD. a=0.003916 A=3.9739E-3 B=Ű5.870E-7 C=Ű4.4E-12.

UeiRTDType3920 Old American standard Platinum RTD. a=0.00392 A=3.9787E-3 B=Ű5.8686E-7 C=Ű4.167E-12.

UeiRTDType3926 ITS-90 standard Platinum RTD. a=0.003926 A=3.9848E-3 B=Ű5.870E-7 C=Ű4.0E-12.

UeiRTDType3928 ITS-90 standard Platinum RTD. a=0.003928 A=3.9888E-3 B=Ű5.915E-7 C=Ű3.85E-12.

UeiRTDTypeCustom Custom RTD. Specify A,B and C using the RTDChannel object attributes.

3.1.3.102 enum _tUeiSensorBridgeType

Sensor bridge configurations

Enumerator:

UeiSensorQuarterBridge Quarter Bridge sensor.

UeiSensorHalfBridge Half Bridge sensor.

UeiSensorFullBridge Full Bridge sensor.

UeiSensorNoBridge No bridge.

3.1.3.103 enum _tUeiSerialPortDataBits

Serial port number of data bits per character frame

Enumerator:

UeiSerialDataBits5 5 bits of data per frame

UeiSerialDataBits6 6 bits of data per frame

UeiSerialDataBits7 7 bits of data per frame

UeiSerialDataBits8 8 bits of data per frame

3.1.3.104 enum _tUeiSerialPortFlowControl

Serial port flow control setting

Enumerator:

UeiSerialFlowControlNone No flow control.

UeiSerialFlowControlRtsCts Hardware flow control, stop sending when CTS is held low, set RTS low when input FIFO is full.

UeiSerialFlowControlXonXoff Software flow control, not implemented yet.

3.1.3.105 enum _tUeiSerialPortMinorFrameMode

Minor framing configures a delay between groups of characters. The mode selects how minor frames are defined

Enumerator:

UeiSerialMinorFrameModeFixedLength Minor frame is a fixed number of characters.

UeiSerialMinorFrameModeZeroChar Minor frame is terminated by a NULL character (0x00).

UeiSerialMinorFrameModeVariableLength Minor frame is prefixed with a byte containing the length of the frame.

3.1.3.106 enum _tUeiSerialPortMode

Serial port mode

Enumerator:

UeiSerialModeRS232 RS-232 mode.
UeiSerialModeRS485HalfDuplex RS-485 Half-Duplex mode.
UeiSerialModeRS485FullDuplex RS-485 Full-Duplex mode.

3.1.3.107 enum _tUeiSerialPortParity

Serial port parity

Enumerator:

UeiSerialParityNone No parity bit.
UeiSerialParityOdd Set parity bit to 1 if the frame contains an even number of bit(s) set to 1.
UeiSerialParityEven Set parity bit to 0 if the frame contains an odd number of bit(s) set to 1.
UeiSerialParityMark Always set parity bit to 1.
UeiSerialParitySpace Always set parity bit to 0.

3.1.3.108 enum _tUeiSerialPortSpeed

Serial port pseed

Enumerator:

UeiSerialBitsPerSecond110 110 bits per second
UeiSerialBitsPerSecond300 300 bits per second
UeiSerialBitsPerSecond600 600 bits per second
UeiSerialBitsPerSecond1200 1200 bits per second
UeiSerialBitsPerSecond2400 2400 bits per second
UeiSerialBitsPerSecond4800 4800 bits per second
UeiSerialBitsPerSecond9600 9600 bits per second
UeiSerialBitsPerSecond14400 14400 bits per second
UeiSerialBitsPerSecond19200 19200 bits per second
UeiSerialBitsPerSecond28800 28800 bits per second
UeiSerialBitsPerSecond38400 38400 bits per second
UeiSerialBitsPerSecond57600 57600 bits per second
UeiSerialBitsPerSecond115200 115200 bits per second
UeiSerialBitsPerSecond128000 128000 bits per second
UeiSerialBitsPerSecond250000 250000 bits per second
UeiSerialBitsPerSecond256000 256000 bits per second (available only in RS-485 mode)
UeiSerialBitsPerSecond1000000 1000000 bits per second (available only in RS-485 mode)
UeiSerialBitsPerSecondCustom Use custom port speed (to be programmed using port attribute).

3.1.3.109 enum _tUeiSerialPortStopBits

Serial port number of stop bits

Enumerator:

UeiSerialStopBits1 Use one stop bit at the end of each frame.
UeiSerialStopBits1_5 Use one and a half stop bits at the end of each frame.
UeiSerialStopBits2 Use two stop bits at the end of each frame.

3.1.3.110 enum _tUeiSerialReadDataType

Data type options for CUeiSerialReader.

Enumerator:

UeiSerialReadDataTypeNoTimestamp No timestamp with data.
UeiSerialReadDataTypeTimestamp Timestamped data.

3.1.3.111 enum _tUeiSessionState

Session state

Enumerator:

UeiSessionStateUnknown Session is in an unknown state.
UeiSessionStateReserved Session has reserved resources.
UeiSessionStateConfigured Session is configured and ready.
UeiSessionStateRunning Session is running.
UeiSessionStateFinished Session is finished.

3.1.3.112 enum _tUeiSessionType

Session type

Enumerator:

UeiSessionTypeAI Analog input session.
UeiSessionTypeAO Analog output session.
UeiSessionTypeDI Port level digital input session.
UeiSessionTypeDO Port level digital output session.
UeiSessionTypeCI Counter input session.
UeiSessionTypeCO Counter output session.
UeiSessionTypeInfo Info retrieval session.
UeiSessionTypeSerial Serial port communication session.
UeiSessionTypeSynchronousSerial IRIG session.
UeiSessionTypeCAN CAN bus communication session.

UeiSessionTypeARINC ARINC-429 bus communication session.

UeiSessionTypeMIL1553 MIL-STD-1553B bus communication session.

UeiSessionTypeIRIG IRIG session.

UeiSessionTypeSync Synchronization session.

UeiSessionTypeDILineLevel Line level digital input session.

UeiSessionTypeDOLineLevel Line level digital output session.

UeiSessionTypeVR Variable reluctance input session.

UeiSessionTypeCSDB CSDB communication session.

UeiSessionTypeSSI SSI communication session.

UeiSessionTypeSPI SPI communication session.

UeiSessionTypeMUX MUX session.

UeiSessionTypeI2C I2C communication session.

UeiSessionTypeDiagnostic Diagnostic input session.

3.1.3.113 enum _tUeiSignalSource

The source of a synchronization signal

Enumerator:

UeiSignalStartTrigger start trigger

UeiSignalStopTrigger stop trigger

UeiSignalScanClock Scan clock signal.

UeiSignalConvertClock Convert Clock signal.

UeiSignalBackplane One of the synchronization lines built-in the back plane.

UeiSignalSoftware A software signal that is emmited with the FireSignal API.

UeiSignalExternal The external sync connector (Only on PowerDNA).

UeiSignalPLL The PLL frequency generator running at multiplied scan rate for devices that require it (Only on PowerDNA).

UeiSignalSync1PPS Clock signal generated by ADPLL and event module.

3.1.3.114 enum _tUeiStrainGageBridgeType

Strain gage bridge configurations

Enumerator:

UeiStrainGageQuarterBridgeI Quarter Bridge I strain gage.

UeiStrainGageQuarterBridgeII Quarter Bridge II strain gage.

UeiStrainGageHalfBridgeI Half Bridge I strain gage.

UeiStrainGageHalfBridgeII Half Bridge II strain gage.

UeiStrainGageFullBridgeI Full Bridge I strain gage.

UeiStrainGageFullBridgeII Full Bridge II strain gage.

UeiStrainGageFullBridgeIII Full Bridge III strain gage.

3.1.3.115 enum _tUeiSync1PPSDataType

Synchronization sessions can read status for different components

Enumerator:

UeiSync1PPSDataFull Read all status values from all components.

UeiSync1PPSDataLocked Read locked status to determine when the event module output clock is stable.

UeiSync1PPSPTPStatus Read PTP status.

UeiSync1PPSPTPUTCTime Read UTC time in seconds since UNIX Epoch Time (01/01/1970).

3.1.3.116 enum _tUeiSync1PPSMode

Specifies the mode used to obtain 1PPS synchronization clock

Enumerator:

UeiSyncClock External 1PPS TTL clock is connected to a sync input pin.

UeiSync1588 1PPS is derived from 1588 master clock

UeiSyncNTP 1PPS is derived from NTP server

UeiSyncIRIG 1PPS is derived from IRIG device

3.1.3.117 enum _tUeiSync1PPSOutput

when synchronizing multiple devices, a master device must output the synchronization 1PPS clock to the slave devices

Enumerator:

UeiSync1PPSNone No output of 1PPS clock.

UeiSync1PPSOutput0 Sync clock is emitted through synchronization output 0.

UeiSync1PPSOutput1 Sync clock is emitted through synchronization output 1.

3.1.3.118 enum _tUeiSync1PPSSource

When in sync clock mode, specifies where the 1PPS clock is coming from

Enumerator:

UeiSync1PPSInternal 1PPS is generated internally

UeiSync1PPSInput0 1PPS is connected to synchronization input 0

UeiSync1PPSInput1 1PPS is connected to synchronization input 1

3.1.3.119 enum _tUeiSync1PPSTriggerOperation

Trigger operations that a 1PPS sync session can use to start slave devices

Enumerator:

UeiSync1PPSTriggerOnNextPPS trigger local slave devices on next PPS rising edge

UeiSync1PPSTriggerOnNextPPSBroadCast trigger local and remote slave devices on next PPS rising edge

3.1.3.120 enum _tUeiSynchroMessageType

Synchro/Resolver message data types (internal use)

Enumerator:

UeiSynchroMessageStageAngles Write angles.

UeiSynchroMessageUpdate Update FIFOs.

3.1.3.121 enum _tUeiSynchroResolverMode

Specifies whether a synchro or a resolver is measured or simulated

Enumerator:

UeiSynchroMode A synchro is measured or simulated.

UeiResolverMode A resolver is measured or simulated.

UeiSynchroZGroundMode A synchro with Z wire connected to ground is measured or simulated.

3.1.3.122 enum _tUeiSyncLine

Specifies the backplane synchronization used to connect the hardware to synchronize

Enumerator:

UeiSyncLine0 Use synchronization 0.

UeiSyncLine1 Use synchronization 1.

UeiSyncLine2 Use synchronization 2.

UeiSyncLine3 Use synchronization 3.

UeiSyncNone Not connected.

3.1.3.123 enum _tUeiTemperatureScale

Temperature scales

Enumerator:

UeiTemperatureScaleCelsius Celsius temperature scale.

UeiTemperatureScaleFahrenheit Fahrenheit temperature scale.

UeiTemperatureScaleKelvin Kelvin temperature scale.

UeiTemperatureScaleRankine Rankine temperature scale.

3.1.3.124 enum _tUeiThermocoupleType

Thermocouple types

Enumerator:

UeiThermocoupleTypeE Type E thermocouple.

UeiThermocoupleTypeJ Type J thermocouple.

UeiThermocoupleTypeK Type K thermocouple.

UeiThermocoupleTypeR Type R thermocouple.

UeiThermocoupleTypeS Type S thermocouple.

UeiThermocoupleTypeT Type T thermocouple.

UeiThermocoupleTypeB Type B thermocouple.

UeiThermocoupleTypeN Type N thermocouple.

UeiThermocoupleTypeC Type C thermocouple.

3.1.3.125 enum _tUeiTimingClockSource

Clock source for a timed operation

Enumerator:

UeiTimingClockSourceInternal Internal hardware clock timing.

UeiTimingClockSourceExternal External clock timing.

UeiTimingClockSourceContinuous Timing clock is continuous and runs at the maximum capability of the hardware.

UeiTimingClockSourceSoftware Software timing.

UeiTimingClockSourceSlave Board synchronization timing.

UeiTimingClockSourceExternalDividedByCounter External clock divided by a counter.

UeiTimingClockSourceSignal The clock is coming from an internal signal.

3.1.3.126 enum _tUeiTimingDuration

Duration of a timed operation

Enumerator:

UeiTimingDurationContinuous Continuous timed operation.

UeiTimingDurationSingleShot One-shot timed operation.

3.1.3.127 enum _tUeiTimingMode

Mode of a timed operation

Enumerator:

UeiTimingModeSimpleIO Simple I/O mode. The data is read/written one scan at a time. It is timed by software.

UeiTimingModeBufferedIO Buffered I/O mode. The data is read/written in buffers. It is timed by a hardware internal or external clock.

UeiTimingModeChangeDetection Digital line state change detection.

UeiTimingModeTimeSequencing Time Sequencing.

UeiTimingModeDirectDataMapping Direct data mapping I/O mode. The data values on inputand output channels are mirrored on the host. It is timed by a hardware internal or external clock.

UeiTimingModeMessagingIO The data is read/written in messages transferred over a communication bus.

UeiTimingModeAsyncIO Asynchronous I/O mode. The data is read/written upon a hardware asynhronous event.

UeiTimingModeAsyncVMapIO lower-level version of ACB, data is pulled directly from device's FIFO without any intermediate buffering

UeiTimingModeVMapIO low-level mode that provides direct access to device's FIFO

3.1.3.128 enum _tUeiTriggerAction

Trigger Action

Enumerator:

UeiTriggerActionStartSession The session will start when the trigger occurs.

UeiTriggerActionStopSession The session will stop when the trigger occurs.

UeiTriggerActionBuffer Each buffer will be acquired or generated when the trigger occurs.

3.1.3.129 enum _tUeiTriggerCondition

Trigger Condition

Enumerator:

UeiTriggerConditionRising The trigger occurs when the signal has a positive slope when passing through the specified value.

UeiTriggerConditionFalling The trigger occurs when the signal has a negative slope when passing through the specified value.

3.1.3.130 enum _tUeiTriggerSource

Trigger Source

Enumerator:

UeiTriggerSourceImmediate Session is triggered as soon as it is started.

UeiTriggerSourceSoftware Session is triggered when a software trigger condition is satisfied.

UeiTriggerSourceExternal Session is triggered by an external signal.

UeiTriggerSourceSignal Session is triggered by a software or internal signal.

UeiTriggerSourceNext1PPS Session is triggered upon the next 1PPS pulse (only valid for synchronization sessions).

UeiTriggerSourceSyncLine0 Session is triggered by backplane synchronization line 0.

UeiTriggerSourceSyncLine1 Session is triggered by backplane synchronization line 1.

UeiTriggerSourceSyncLine2 Session is triggered by backplane synchronization line 2.

UeiTriggerSourceSyncLine3 Session is triggered by backplane synchronization line 3.

3.1.3.131 enum _tUeiVRAPTMode

Configures the adaptive peak threshold (APT) mode on variable reluctance sessions. APT finds the point in time where the VR sensor output voltage falls below a certain threshold. This point marks the beginning of the gap between two teeth.

Enumerator:

UeiAPTModeChip The front-end IC will automatically set the AP threshold to 1/3 of the peak input voltage.

UeiAPTModeLogic The device's FPGA measures the VR sensor signal and sets the AP threshold to a programmable fraction of the peak input voltage.

UeiAPTModeFixed Turn-off APT and use hard-coded AP threshold.

UeiAPTModeTTL Turn-off APT, use this mode when connecting a TTL signal to the VR input.

3.1.3.132 enum _tUeiVRDataType

VR sessions can read single measurements, FIFO based data, and AI FIFO data

Enumerator:

UeiVRDataPositionVelocity Read position and velocity measurement.

UeiVRDataFifoPosition Read multiple position measurements. This is useful to calculate inter-tooth timing and acceleration.

UeiVRDataADCFifo Read ADC FIFO. this is useful for diagnostic purpose.

UeiVRDataADCStatus Read ADC status.

3.1.3.133 enum _tUeiVRDigitalSource

The source of a digital output on a variable reluctance device

Enumerator:

UeiVRDigitalSourceDisable Disable output.

UeiVRDigitalSourceForceHigh Set output level to high.

UeiVRDigitalSourceForceLow Set output level to low.

UeiVRDigitalSourceZTooth7 Set output to high when Z tooth is detected on channel 7.

UeiVRDigitalSourceZTooth6 Set output to high when Z tooth is detected on channel 6.

UeiVRDigitalSourceZTooth5 Set output to high when Z tooth is detected on channel 5.

UeiVRDigitalSourceZTooth4 Set output to high when Z tooth is detected on channel 4.

UeiVRDigitalSourceZTooth3 Set output to high when Z tooth is detected on channel 3.

UeiVRDigitalSourceZTooth2 Set output to high when Z tooth is detected on channel 2.

UeiVRDigitalSourceZTooth1 Set output to high when Z tooth is detected on channel 1.

UeiVRDigitalSourceZTooth0 Set output to high when Z tooth is detected on channel 0.

UeiVRDigitalSourceNPulse7 Set output to high when N teeth have been detected on channel 7.

UeiVRDigitalSourceNPulse6 Set output to high when N teeth have been detected on channel 6.

UeiVRDigitalSourceNPulse5 Set output to high when N teeth have been detected on channel 5.

UeiVRDigitalSourceNPulse4 Set output to high when N teeth have been detected on channel 4.

UeiVRDigitalSourceNPulse3 Set output to high when N teeth have been detected on channel 3.

UeiVRDigitalSourceNPulse2 Set output to high when N teeth have been detected on channel 2.

UeiVRDigitalSourceNPulse1 Set output to high when N teeth have been detected on channel 1.

UeiVRDigitalSourceNPulse0 Set output to high when N teeth have been detected on channel 0.

3.1.3.134 enum _tUeiVRFIFOMode

Variable reluctance FIFO data can be used to calculate a map of the inter-tooth delays around the encoder wheel. this is useful to calculate acceleration.

Enumerator:

UeiFIFOModeDisabled No data is pushed to FIFO.

UeiFIFOModePosition Raw counter data is pushed to FIFO.

UeiFIFOModePosAndTS Raw counter data and timestamp is pushed to FIFO.

3.1.3.135 enum _tUeiVRMode

Configures the input mode on variable reluctance sessions. Variable reluctance input device converts the VR sensor analog signal to a digital signal that can be processed by a counter/timer The VR-608 can use four different modes to measure velocity, position or direction The counting mode can be set to: Decoder: Even and Odd channels are used in pair to determine direction and position Timed: Count number of teeth detected during a timed interval N pulses: Measure the time taken to detect N teeth (Number of teeth needs to be set separately) Z pulse: Measure the number of teeth and the time elapsed between two Z pulses (The Z tooth is usually a gap or a double tooth on the encoder wheel)

Enumerator:

UeiVRModeDecoder Even and Odd channels are used in pair to measure direction and position.

UeiVRModeCounterTimed Count number of teeth detected during given time interval to measure velocity and position.

UeiVRModeCounterNPulses Count time taken to detect N teeth to measure velocity only.

UeiVRModeCounterZPulse Count elapsed time and number of teeth detected between two Z pulses to measure velocity and position.

3.1.3.136 enum _tUeiVRZCMode

Configures the zero-crossing mode on variable reluctance sessions. Zero crossing finds the point in time where the VR sensor output voltage goes from positive to negative voltage. This point is the center of the tooth lining up with the center of the VR sensor

Enumerator:

UeiZCModeChip The front-end IC will automatically calculate the ZC level.

UeiZCModeLogic The device's FPGA measures the VR sensor signal and calculate the ZC level as (min+max)/2.

UeiZCModeFixed Use hard-coded ZC level.

3.1.3.137 enum _tUeiWatchDogCommand

Configures the watchdog timer to reset the device it is attached to when a programmable timer delay expires. Periodically clearing the timer prevents reset.

Enumerator:

UeiWatchDogDisable Disable watchdog.

UeiWatchDogEnableClearOnReceive Configure watchdog to clear timer upon receiving a packet.

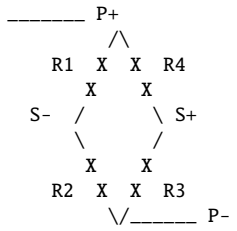
UeiWatchDogEnableClearOnTransmit Configure watchdog to clear timer upon transmitting a packet.

UeiWatchDogEnableClearOnCommand Configure watchdog to clear timer upon receiving a clear command.

UeiWatchDogClearTimer Clear the timer.

3.1.3.138 enum _tUeiWheatstoneBridgeBranch

Specify one of the four branches on a Wheatstone bridge



Enumerator:

UeiWheatstoneBridgeR1 R1 branch (P+ S-).

UeiWheatstoneBridgeR2 R2 branch (S- P-).

UeiWheatstoneBridgeR3 R3 branch (S+ P-).

UeiWheatstoneBridgeR4 R4 branch (P+ S+).

3.1.3.139 enum _tUeiWiringScheme

Sensor with excitation wiring schemes

Enumerator:

UeiTwoWires Sensor is connected using two wires.

UeiFourWires Sensor is connected using four wires.

UeiSixWires Sensor is connected using six wires.

UeiThreeWires Sensor is connected using three wires.

3.2 UeiDaqAnsiC.h File Reference

Typedefs

- typedef void * **SessionHandle**
  The SessionHandle type represents a session object.
- typedef void * **DeviceHandle**
  The DeviceHandle type represents a device object.
- typedef void * **TimingHandle**
  The TimingHandle type represents a timing object.
- typedef void * **TriggerHandle**
  The TriggerHandle type represents a trigger object.
- typedef void * **ChannelHandle**
  The ChannelHandle type represents a channel object.
- typedef void * **DataStreamHandle**
  The DataStreamHandle type represents a DataStream object.
- typedef void(*) **tUeiEventCallback** (**tUeiEvent** event, void *param)
  Asynchronous callback function prototype.

Functions

- UeiDaqAPI int **UeiDaqCreateSession** (**SessionHandle** *sessionHandle)
  Create a session.
- UeiDaqAPI int **UeiDaqCloseSession** (**SessionHandle** sessionHandle)
  Close a session.
- UeiDaqAPI int **UeiDaqCloseDevice** (**DeviceHandle** deviceHandle)
  Close a device.
- UeiDaqAPI int **UeiDaqCreateDiagnosticChannel** (**SessionHandle** sessionHandle, char *resource)
  Create Diagnostic input channel list.
- UeiDaqAPI int **UeiDaqCreateAIChannel** (**SessionHandle** sessionHandle, char *resource, **f64** min, **f64** max, **tUeiAIChannelInputMode** mode)
  Create Analog voltage input channel list.
- UeiDaqAPI int **UeiDaqCreateAICurrentChannel** (**SessionHandle** sessionHandle, char *resource, **f64** min, **f64** max, **tUeiFeatureEnable** enableCB, **tUeiAIChannelInputMode** mode)
  Create Analog current input channel list.

- UeiDaqAPI int **UeiDaqCreateAOChannel** (**SessionHandle** sessionHandle, char *resource, **f64** min, **f64** max)

  Create Analog output channel list.

- UeiDaqAPI int **UeiDaqCreateAOCurrentChannel** (**SessionHandle** sessionHandle, char *resource, **f64** min, **f64** max)

  Create Analog output current channel list.

- UeiDaqAPI int **UeiDaqCreateAOWaveformChannel** (**SessionHandle** sessionHandle, char *resource, **tUeiAOWaveformClockSource** dacClockSource, **tUeiAOWaveformOffsetClockSource** offsetDACClockSource, **tUeiAOWaveformClockSync** clockSync)

  Create Analog output waveform channel list.

- UeiDaqAPI int **UeiDaqCreateAOProtectedChannel** (**SessionHandle** sessionHandle, char *resource, **tUeiAODACMode** mode, double measurementRate, int autoRetry, double retryRate)

  Create Protected analog output channel list.

- UeiDaqAPI int **UeiDaqCreateAOProtectedCurrentChannel** (**SessionHandle** sessionHandle, char *resource, **tUeiAODACMode** mode, double measurementRate, int autoRetry, double retryRate)

  Create Protected analog output current channel list.

- UeiDaqAPI int **UeiDaqCreateSimulatedTCChannel** (**SessionHandle** sessionHandle, char *resource, **tUeiThermocoupleType** tcType, **tUeiTemperatureScale** tempScale, int enableCjc)

  Create Simulated Thermocouple output channel list.

- UeiDaqAPI int **UeiDaqCreateSimulatedRTDChannel** (**SessionHandle** sessionHandle, char *resource, **tUeiRTDType** rtdType, double rtdNominalResistance, **tUeiTemperatureScale** tempScale)

  Create simulated RTD output channel list.

- UeiDaqAPI int **UeiDaqCreateDIChannel** (**SessionHandle** sessionHandle, char *resource)

  Create Digital input channel list.

- UeiDaqAPI int **UeiDaqCreateDOChannel** (**SessionHandle** sessionHandle, char *resource)

  Create Digital input channel list.

- UeiDaqAPI int **UeiDaqCreateCIChannel** (**SessionHandle** sessionHandle, char *resource, **tUeiCounterSource** source, **tUeiCounterMode** mode, **tUeiCounterGate** gate, **Int32** divider, **Int32** inverted)

  Create Counter input channel list.

- UeiDaqAPI int **UeiDaqCreateVRChannel** (**SessionHandle** sessionHandle, char *resource, **tUeiVRMode** mode)

  Create Variable Reluctance input channel list.

- UeiDaqAPI int **UeiDaqCreateQuadratureEncoderChannel** (**SessionHandle** sessionHandle, char *resource, **uInt32** initialPosition, **tUeiQuadratureDecodingType** decodingType, **Int32** enableZeroIndexing, **tUeiQuadratureZeroIndexPhase** zeroIndexPhase)

  Create quadrature encoder input channel list.

- UeiDaqAPI int **UeiDaqCreateCOChannel** (**SessionHandle** sessionHandle, char *resource, **tUeiCounterSource** source, **tUeiCounterMode** mode, **tUeiCounterGate** gate, **uInt32** tick1, **uInt32** tick2, **Int32** divider, **Int32** inverted)

  Create Counter output channel list.

- UeiDaqAPI int **UeiDaqCreateTCChannel** (**SessionHandle** sessionHandle, char *resource, **f64** min, **f64** max, **tUeiThermocoupleType** tcType, **tUeiTemperatureScale** tempScale, **tUeiColdJunctionCompensationType** cjcType, **f64** cjcConstant, char *cjcResource, **tUeiAIChannelInputMode** mode)

  Create Thermocouple input channel list.

- UeiDaqAPI int **UeiDaqCreateResistanceChannel** (**SessionHandle** sessionHandle, char *resource, **f64** min, **f64** max, **tUeiWiringScheme** wiring, double twoWiresLeadResistance, **tUeiAIChannelInputMode** mode)

  Create Resistance input channel list.

- UeiDaqAPI int **UeiDaqCreateRTDChannel** (**SessionHandle** sessionHandle, char *resource, **f64** min, **f64** max, **tUeiWiringScheme** wiring, double twoWiresLeadResistance, **tUeiRTDType** rtdType, double rtdNominalResistance, **tUeiTemperatureScale** tempScale, **tUeiAIChannelInputMode** mode)

  Create RTD input channel list.

- UeiDaqAPI int **UeiDaqCreateAIVExChannel** (**SessionHandle** sessionHandle, char *resource, **f64** min, **f64** max, **tUeiSensorBridgeType** sensorBridgeType, double excitationVoltage, int useExcitationForScaling, **tUeiAIChannelInputMode** mode)

  Create Analog input with excitation channel list.

- UeiDaqAPI int **UeiDaqCreateAccelChannel** (**SessionHandle** sessionHandle, char *resource, **f64** min, **f64** max, double sensorSensitivity, double excitationCurrent, **tUeiCoupling** coupling, int enableLowPassFilter)

  Create Accelerometer channel list.

- UeiDaqAPI int **UeiDaqCreateDMMChannel** (**SessionHandle** sessionHandle, char *resource, **f64** range, **tUeiDMMMeasurementMode** measurementMode)

  Create digital multi-meter channel.

- UeiDaqAPI int **UeiDaqCreateLVDTChannel** (**SessionHandle** sessionHandle, char *resource, **f64** min, **f64** max, double sensorSensitivity, **tUeiLVDTWiringScheme** wiringScheme, double excitationVoltage, double excitationFrequency, int externalExcitation)

  Create LVDT/RVDT input channel list.

- UeiDaqAPI int **UeiDaqCreateSimulatedLVDTChannel** (**SessionHandle** sessionHandle, char *resource, double sensorSensitivity, **tUeiLVDTWiringScheme** wiringScheme, double excitationVoltage, double excitationFrequency)

  Create simulated LVDT/RVDT output channel list.

- UeiDaqAPI int **UeiDaqCreateSynchroResolverChannel** (**SessionHandle** sessionHandle, char *resource, **tUeiSynchroResolverMode** mode, double excitationVoltage, double excitationFrequency, int externalExcitation)

  Create Synchro/Resolver analog input channel list.

- UeiDaqAPI int **UeiDaqCreateSimulatedSynchroResolverChannel** (**SessionHandle** sessionHandle, char *resource, **tUeiSynchroResolverMode** mode, double excitationVoltage, double excitationFrequency, int externalExcitation)

  Create simulated Synchro/Resolver analog output channel list.

- UeiDaqAPI int **UeiDaqCreateSerialPort** (**SessionHandle** sessionHandle, char *resource, **tUeiSerialPortMode** mode, **tUeiSerialPortSpeed** bitsPerSecond, **tUeiSerialPortDataBits** dataBits, **tUeiSerialPortParity** parity, **tUeiSerialPortStopBits** stopBits, char *termination)

  Create serial port.

- UeiDaqAPI int **UeiDaqCreateHDLCPort** (**SessionHandle** sessionHandle, char *resource, **tUeiHDLCPortPhysical** physInterface, **uInt32** bitsPerSecond, **tUeiHDLCPortEncoding** encoding, **tUeiHDLCPortCRCMode** crcMode, **tUeiHDLCPortClockSource** txClockSource, **tUeiHDLCPortClockSource** rxClockSource)

  Create HDLC port.

- UeiDaqAPI int **UeiDaqCreateCANPort** (**SessionHandle** sessionHandle, char *resource, **tUeiCANPortSpeed** bitsPerSecond, **tUeiCANFrameFormat** frameFormat, **tUeiCANPortMode** mode, **uInt32** acceptanceMask, **uInt32** acceptanceCode)

  Create CAN port.

- UeiDaqAPI int **UeiDaqCreateDOIndustrialChannel** (**SessionHandle** sessionHandle, char *resource, **tUeiDOPWMMode** pwmMode, **uInt32** pwmLengthUs, **uInt32** pwmPeriodUs, double pwmDutyCycle)

  Create Industrial digital output channel list.

- UeiDaqAPI int **UeiDaqCreateDOProtectedChannel** (**SessionHandle** sessionHandle, char *resource, double underCurrentLimit, double overCurrentLimit, double currentSampleRate, int autoRetry, double retryRate)

  Create Protected digital output channel list.

- UeiDaqAPI int **UeiDaqCreateDIIndustrialChannel** (**SessionHandle** sessionHandle, char *resource, double lowThreshold, double highThreshold, double minPulseWidth)

  Create industrial digital input channel list.

- UeiDaqAPI int **UeiDaqCreateARINCInputPort** (**SessionHandle** sessionHandle, char *resource, **tUeiARINCPortSpeed** bitsPerSecond, **tUeiARINCPortParity** parity, int enableSDIFilter, **uInt32** SDIMask)

  Create ARINC input port.

- UeiDaqAPI int **UeiDaqCreateARINCOutputPort** (**SessionHandle** sessionHandle, char *resource, **tUeiARINCPortSpeed** bitsPerSecond, **tUeiARINCPortParity** parity)

  Create ARINC output port.

- UeiDaqAPI int **UeiDaqCreateMIL1553Port** (**SessionHandle** sessionHandle, char *resource, **tUeiMIL1553PortCoupling** coupling, **tUeiMIL1553PortOpMode** portMode)

Create MIL-1553 port.

- UeiDaqAPI int **UeiDaqCreateIRIGTimeKeeperChannel** (**SessionHandle** sessionHandle, char *resource, **tUeiIRIGTimeKeeper1PPSSource** source, int autoFollow)

  Create IRIG time keeper channel.

- UeiDaqAPI int **UeiDaqCreateIRIGInputChannel** (**SessionHandle** sessionHandle, char *resource, **tUeiIRIGDecoderInputType** inputType, **tUeiIRIGTimeCodeFormat** timeCodeFormat)

  Create IRIG input channel.

- UeiDaqAPI int **UeiDaqCreateIRIGOutputChannel** (**SessionHandle** sessionHandle, char *resource, **tUeiIRIGTimeCodeFormat** timeCodeFormat)

  Create IRIG output channel.

- UeiDaqAPI int **UeiDaqCreateIRIGDOTTLChannel** (**SessionHandle** sessionHandle, char *resource, **tUeiIRIGDOTTLSource** source0, **tUeiIRIGDOTTLSource** source1, **tUeiIRIGDOTTLSource** source2, **tUeiIRIGDOTTLSource** source3)

  Create IRIG DO TTL channel.

- UeiDaqAPI int **UeiDaqCreateSync1PPSPort** (**SessionHandle** sessionHandle, char *resource, **tUeiSync1PPSMode** mode, **tUeiSync1PPSSource** source, double EMOutputRate)

  Create 1PPS synchronization port.

- UeiDaqAPI int **UeiDaqCreateCSDBPort** (**SessionHandle** sessionHandle, char *resource, int bps, int parity, int blockSize, int numberOfMessages, int interByteDelayUs, int interBlockDelayUs, int framePeriodUs)

  Create CSDB port.

- UeiDaqAPI int **UeiDaqCreateSSIMasterPort** (**SessionHandle** sessionHandle, char *resource, unsigned int bps, unsigned int wordSize, int enableClock, int enableTermination, double pauseTime, double transferTimeout, double bitUpdateTime)

  Create SSI master port list.

- UeiDaqAPI int **UeiDaqCreateSSISlavePort** (**SessionHandle** sessionHandle, char *resource, unsigned int bps, unsigned int wordSize, int enableTransmit, int enableTermination, double pauseTime, double transferTimeout, double bitUpdateTime)

  Create SSI slave port list.

- UeiDaqAPI int **UeiDaqCreateMuxPort** (**SessionHandle** sessionHandle, char *resource, int breakBeforeMake)

  Create Mux port.

- UeiDaqAPI int **UeiDaqCreateI2CSlavePort** (**SessionHandle** sessionHandle, char *resource, **tUeiI2CTTLLevel** ttlLevel, **tUeiI2CSlaveAddressWidth** addressWidth, int address)

  Create I2C slave port.

- UeiDaqAPI int **UeiDaqCreateI2CMasterPort** (**SessionHandle** sessionHandle, char *resource, **tUeiI2CPortSpeed** bitRate, **tUeiI2CTTLLevel** ttlLevel, int enableSecureShell)

  Create I2C master port.

- UeiDaqAPI int **UeiDaqConfigureTimingForSimpleIO** (**SessionHandle** sessionHandle)

  Configure the timing object for simple IO.

- UeiDaqAPI int **UeiDaqConfigureTimingForBufferedIO** (**SessionHandle** sessionHandle, int samplePerChannel, **tUeiTimingClockSource** clkSource, double rate, **tUeiDigitalEdge** edge, **tUeiTimingDuration** duration)

  Configure the timing object for buffered mode.

- UeiDaqAPI int **UeiDaqConfigureTimingForEdgeDetection** (**SessionHandle** sessionHandle, **tUeiDigitalEdge** edge)

  Configure the timing object for digital line state change detection.

- UeiDaqAPI int **UeiDaqConfigureTimingForAsynchronousIO** (**SessionHandle** sessionHandle, int watermark, int periodUs, int noActivityTimeoutUs, int maxDataSize)

  Configure the timing object for asynchronous FIFO based I/O.

- UeiDaqAPI int **UeiDaqConfigureTimingForTimeSequencing** (**SessionHandle** sessionHandle, int samplePerChannel, **tUeiTimingDuration** duration)

  Configure the timing object for time sequencing.

- UeiDaqAPI int **UeiDaqConfigureTimingForDataMappingIO** (**SessionHandle** sessionHandle, **tUeiTimingClockSource** clkSource, double rate)

  Configure the timing object for data map mode.

- UeiDaqAPI int **UeiDaqConfigureTimingForMessagingIO** (**SessionHandle** sessionHandle, int bufferSize, double refreshRate)

  Configure the timing object for messaging IO.

- UeiDaqAPI int **UeiDaqConfigureTimingForVMapIO** (**SessionHandle** sessionHandle, double rate)

  Configure the timing object for variable map mode.

- UeiDaqAPI int **UeiDaqConfigureStartDigitalTrigger** (**SessionHandle** sessionHandle, **tUeiTriggerSource** source, **tUeiDigitalEdge** edge)

  Configure the digital trigger condition that will start the session.

- UeiDaqAPI int **UeiDaqConfigureStopDigitalTrigger** (**SessionHandle** sessionHandle, **tUeiTriggerSource** source, **tUeiDigitalEdge** edge)

  Configure the digital trigger condition that will stop the session.

- UeiDaqAPI int **UeiDaqConfigureAnalogSoftwareTrigger** (**SessionHandle** sessionHandle, **tUeiTriggerAction** action, **tUeiTriggerCondition** condition, **Int32** triggerChannel, **f64** level, **f64** hysteresis, **Int32** preTriggerScans)

  Configure the analog software trigger.

- UeiDaqAPI int **UeiDaqConfigureSignalTrigger** (**SessionHandle** sessionHandle, **tUeiTriggerAction** action, char *signal)

  Configure the trigger signal that will start or stop the session.

- UeiDaqAPI int **UeiDaqStartSession** (**SessionHandle** sessionHandle)

Start the session.

- UeiDaqAPI int **UeiDaqIsSessionRunning** (**SessionHandle** sessionHandle, int *done)

  Check session status.

- UeiDaqAPI int **UeiDaqWaitUntilSessionIsDone** (**SessionHandle** sessionHandle, int timeout, int *done)

  Wait for the specified time until the session is stopped.

- UeiDaqAPI int **UeiDaqStopSession** (**SessionHandle** sessionHandle)

  Stop the session.

- UeiDaqAPI int **UeiDaqPauseSession** (**SessionHandle** sessionHandle, int port, **uInt32** *numValuesPending)

  Pause one port in the session.

- UeiDaqAPI int **UeiDaqResumeSession** (**SessionHandle** sessionHandle, int port)

  Pause one port in the session.

- UeiDaqAPI int **UeiDaqCleanUpSession** (**SessionHandle** sessionHandle)

  Clean-up the session object.

- UeiDaqAPI int **UeiDaqEnableSessionAutoReStart** (**SessionHandle** sessionHandle, int enable)

  Enable/Disable session auto-restart.

- UeiDaqAPI int **UeiDaqGetDeviceHandle** (**SessionHandle** sessionHandle, **DeviceHandle** *device)

  Get a handle to the device object.

- UeiDaqAPI int **UeiDaqGetTimingHandle** (**SessionHandle** sessionHandle, **TimingHandle** *timing)

  Get a handle to the timing object.

- UeiDaqAPI int **UeiDaqGetDataStreamHandle** (**SessionHandle** sessionHandle, **DataStreamHandle** *stream)

  Get a handle to the stream object.

- UeiDaqAPI int **UeiDaqGetStartTriggerHandle** (**SessionHandle** sessionHandle, **TriggerHandle** *trigger)

  Get a handle to the start trigger object.

- UeiDaqAPI int **UeiDaqGetStopTriggerHandle** (**SessionHandle** sessionHandle, **TriggerHandle** *trigger)

  Get a handle to the stop trigger object.

- UeiDaqAPI int **UeiDaqGetNumberOfChannels** (**SessionHandle** sessionHandle, int *numChannels)

  Get the number of channels.

- UeiDaqAPI int **UeiDaqGetChannelHandle** (**SessionHandle** sessionHandle, int index, **ChannelHandle** *channel)

  Get the handle on a channel object.

- UeiDaqAPI int **UeiDaqGetChannelHandlelById** (**SessionHandle** sessionHandle, int id, **ChannelHandle** *channel)

  Get the handle on a channel object by physical channel id.

- UeiDaqAPI int **UeiDaqReadRawData16** (**SessionHandle** sessionHandle, **Int32** timeout, **Int32** numScans, **uInt16** *pBuffer)

  Read 16 bits wide raw scans from the stream.

- UeiDaqAPI int **UeiDaqReadRawData16Async** (**SessionHandle** sessionHandle, **Int32** numScans, **uInt16** *pBuffer, **tUeiEventCallback** pEventCallback)

  Read 16 bits wide raw scans asynchronously from the stream.

- UeiDaqAPI int **UeiDaqReadRawData32** (**SessionHandle** sessionHandle, **Int32** timeout, **Int32** numScans, **uInt32** *pBuffer)

  Read 32 bits wide raw scans from the stream.

- UeiDaqAPI int **UeiDaqReadRawData32Async** (**SessionHandle** sessionHandle, **Int32** numScans, **uInt32** *pBuffer, **tUeiEventCallback** pEventCallback)

  Read 32 bits wide raw scans asynchronously from the stream.

- UeiDaqAPI int **UeiDaqReadScaledData** (**SessionHandle** sessionHandle, **Int32** timeout, **Int32** numScans, **f64** *pBuffer)

  Read scaled scans from the stream.

- UeiDaqAPI int **UeiDaqReadScaledDataAsync** (**SessionHandle** sessionHandle, **Int32** numScans, **f64** *pBuffer, **tUeiEventCallback** pEventCallback)

  Read scaled scans asynchronously from the stream.

- UeiDaqAPI int **UeiDaqReadMessage** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numBytes, void *pBuffer, **Int32** *numBytesRead)

  Read message from the stream.

- UeiDaqAPI int **UeiDaqReadMessageAsync** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numBytes, void *pBuffer, **tUeiEventCallback** pEventCallback)

  Read messages asynchronously from the stream.

- UeiDaqAPI int **UeiDaqReadSerialMessageTimestamped** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numBytes, void *pBuffer, **Int32** *numBytesRead)

  Read serial message from the stream with timestamp.

- UeiDaqAPI int **UeiDaqReadSerialMessageTimestampedAsync** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numBytes, void *pBuffer, **tUeiEventCallback** pEventCallback)

  Read serial messages asynchronously from the stream with timestamp.

- UeiDaqAPI int **UeiDaqReadCANFrame** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numFrames, **tUeiCANFrame** *pBuffer, **Int32** *numFramesRead)

  Read CAN frames from the stream.

- UeiDaqAPI int **UeiDaqReadCANFrameAsync** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numFrames, **tUeiCANFrame** *pBuffer, **tUeiEventCallback** pEventCallback)

  Read CAN frames asynchronously from the stream.

- UeiDaqAPI int **UeiDaqReadARINCWords** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numWords, **tUeiARINCWord** *pBuffer, **Int32** *numWordsRead)

  Read ARINC words from the stream.

- UeiDaqAPI int **UeiDaqReadARINCRawWords** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numWords, **uInt32** *pBuffer, **Int32** *numWordsRead)

  Read ARINC raw words from the stream.

- UeiDaqAPI int **UeiDaqReadARINCWordsAsync** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numWords, **tUeiARINCWord** *pBuffer, **tUeiEventCallback** pEventCallback)

  Read ARINC words asynchronously from the stream.

- UeiDaqAPI int **UeiDaqReadMIL1553Frames** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numFrames, **tUeiMIL1553Frame** *pBuffer, **Int32** *numFramesRead)

  Read MIL-1553 frames from the stream.

- UeiDaqAPI int **UeiDaqReadMIL1553BMFrames** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numFrames, **tUeiMIL1553BMFrame** *pBuffer, **Int32** *numFramesRead)

  Read MIL-1553 bus monitor frames from the stream.

- UeiDaqAPI int **UeiDaqReadMIL1553BMCmdFrames** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numFrames, **tUeiMIL1553BMCmdFrame** *pBuffer, **Int32** *numFramesRead)

  Read MIL-1553 RT status frames from the stream.

- UeiDaqAPI int **UeiDaqReadMIL1553RTDataFrames** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numFrames, **tUeiMIL1553RTFrame** *pBuffer, **Int32** *numFramesRead)

  Read MIL-1553 RT data area frames from the stream.

- UeiDaqAPI int **UeiDaqReadMIL1553RTStatusFrames** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numFrames, **tUeiMIL1553RTStatusFrame** *pBuffer, **Int32** *numFramesRead)

  Read MIL-1553 RT status frames from the stream.

- UeiDaqAPI int **UeiDaqReadMIL1553BCCBStatusFrames** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numFrames, **tUeiMIL1553BCCBStatusFrame** *pBuffer, **Int32** *numFramesRead)

  Read MIL-1553 BCCB status frames (result of entry and data for Tx command).

- UeiDaqAPI int **UeiDaqReadMIL1553BCSchedFrames** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numFrames, **tUeiMIL1553BCSchedFrame** *pBuffer, **Int32** *numFramesRead)

  Read MIL-1553 BCCB status frames (result of entry and data for Tx command).

- UeiDaqAPI int **UeiDaqReadMIL1553BCStatus** (**SessionHandle** sessionHandle, **Int32** port, **tUeiMIL1553BCStatusFrame** *pBuffer)

Read MIL-1553 bus controller status (result of entry and data for Tx command).

- UeiDaqAPI int **UeiDaqReadMIL1553FramesAsync** (**SessionHandle** sessionHandle, **Int32** port, **Int32** bufferSize, **tUeiMIL1553Frame** *pBuffer, **tUeiEventCallback** pEventCallback)

  Read MIL-1553 frames from the stream.

- UeiDaqAPI int **UeiDaqReadMIL1553A708DataFrames** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numFrames, **tUeiMIL1553A708DataFrame** *pBuffer, **Int32** *numFramesRead)

  Read ARINC 708 data frames from the stream.

- UeiDaqAPI int **UeiDaqReadIRIGTimeKeeperBCDTime** (**SessionHandle** sessionHandle, **Int32** port, **tUeiBCDTime** *pBuffer, **Int32** *status)

  Read current time in BCD format.

- UeiDaqAPI int **UeiDaqReadIRIGTimeKeeperSBSTime** (**SessionHandle** sessionHandle, **Int32** port, **tUeiSBSTime** *pBuffer, **Int32** *status)

  Read current time in SBS format.

- UeiDaqAPI int **UeiDaqReadIRIGTimeKeeperCANSITime** (**SessionHandle** sessionHandle, **Int32** port, **tUeiANSITime** *pBuffer, **Int32** *status)

  Read current time in C-ANSI format.

- UeiDaqAPI int **UeiDaqReadIRIGGPS** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numBytes, char *pBuffer, **Int32** *numBytesRead)

  Read NMEA data from GPS.

- UeiDaqAPI int **UeiDaqReadIRIGTReg** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numRegisters, **uInt32** *pBuffer, **Int32** *numRegistersRead)

  Read time registers.

- UeiDaqAPI int **UeiDaqReadIRIGRawTimeCode** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numTimeCodeChars, **uInt32** *pBuffer, **Int32** *numTimeCodeCharsRead)

  Read raw input time code.

- UeiDaqAPI int **UeiDaqReadHDLCFrame** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numBytes, void *pBuffer, **Int32** *numBytesRead)

  Read HDLC frame from the stream.

- UeiDaqAPI int **UeiDaqReadVRData** (**SessionHandle** sessionHandle, **Int32** channel, **Int32** numValues, **tUeiVRData** *pBuffer, **Int32** *numValuesRead)

  Read variable reluctance data.

- UeiDaqAPI int **UeiDaqReadVRFifoData** (**SessionHandle** sessionHandle, **Int32** channel, **Int32** numValues, **uInt32** *pBuffer, **Int32** *numValuesRead)

  Read variable reluctance FIFO data.

- UeiDaqAPI int **UeiDaqReadVRADCData** (**SessionHandle** sessionHandle, **Int32** channel, **Int32** numValues, double *pBuffer, **Int32** *numValuesRead)

  Read variable reluctance ADC data.

- UeiDaqAPI int **UeiDaqReadVRADCStatus** (**SessionHandle** sessionHandle, **Int32** channel, **uInt32** *pStatus)

  Read variable reluctance ADC status.

- UeiDaqAPI int **UeiDaqReadCSDBFrame** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numMessages, **tUeiCSDBMessage** *pBuffer, **Int32** *numMessagesRead)

  Read CSDB message blocks from the stream.

- UeiDaqAPI int **UeiDaqReadSync1PPSStatus** (**SessionHandle** sessionHandle, **tUeiSync1PPSStatus** *pBuffer)

  Read 1PPS synchronization status from the stream.

- UeiDaqAPI int **UeiDaqReadSync1PPSLockedStatus** (**SessionHandle** sessionHandle, int *isLocked)

  Read 1PPS synchronization locked status from the stream.

- UeiDaqAPI int **UeiDaqReadSync1PPSPTPStatus** (**SessionHandle** sessionHandle, **tUeiSync1PPSPTPStatus** *pBuffer)

  Read PTP status from the stream.

- UeiDaqAPI int **UeiDaqReadSync1PPSPTPUTCTime** (**SessionHandle** sessionHandle, **tUeiPTPTime** *pBuffer)

  Read PTP UTC time from the stream.

- UeiDaqAPI int **UeiDaqReadSSIMasterWords** (**SessionHandle** sessionHandle, int port, int grayDecoding, **Int32** numWords, **uInt32** *pBuffer, **Int32** *numWordsRead)

  Read word(s) from the SSI master port.

- UeiDaqAPI int **UeiDaqReadMuxStatus** (**SessionHandle** sessionHandle, **uInt32** *relayA, **uInt32** *relayB, **uInt32** *relayC, **uInt32** *status)

  Get Mux status.

- UeiDaqAPI int **UeiDaqReadMuxADC** (**SessionHandle** sessionHandle, **Int32** numVals, double *pBuffer)

  Read ADC.

- UeiDaqAPI int **UeiDaqReadMuxReadRelayCounts** (**SessionHandle** sessionHandle, **Int32** countsBufferSize, **Int32** *pCountsBuffer, **Int32** *numCountsRead)

  Read current count of times each relay has been energized.

- UeiDaqAPI int **UeiDaqReadLVDTCoilAmplitudes** (**SessionHandle** sessionHandle, **f64** *pPrimaryCoilsBuffer, **f64** *pSecondaryCoilsBuffer)

  Read primary and secondary coils RMS voltages.

- UeiDaqAPI int **UeiDaqReadI2CSlave** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numMessages, **tUeiI2CSlaveMessage** *pBuffer, **Int32** *numMessagesRead)

  Read messages received by slave.

- UeiDaqAPI int **UeiDaqReadI2CMaster** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numMessages, **tUeiI2CMasterMessage** *pBuffer, **Int32** *numMessagesRead)

  Read messages received by master.

- UeiDaqAPI int **UeiDaqReadDMMStatus** (**SessionHandle** sessionHandle, **uInt32** *pStatus)

  Read DMM status returned with last read data.

- UeiDaqAPI int **UeiDaqIsDMMProtectionTripped** (**SessionHandle** sessionHandle, int *isTripped)

  Read DMM protection tripped status.

- UeiDaqAPI int **UeiDaqIsDMMProtectionTrippedMaxRange** (**SessionHandle** sessionHandle, int *isTrippedMax)

  Read DMM protection tripped in max range status.

- UeiDaqAPI int **UeiDaqIsDMMProtectionReconfiguredSafeRange** (**SessionHandle** sessionHandle, int *isReconfigured)

  Read DMM protection reconfigured to safe range status.

- UeiDaqAPI int **UeiDaqWriteRawData16** (**SessionHandle** sessionHandle, **Int32** timeout, **Int32** numScans, **uInt16** *pBuffer)

  Write 16-bit wide raw scans to the stream.

- UeiDaqAPI int **UeiDaqWriteRawData16Async** (**SessionHandle** sessionHandle, **Int32** numScans, **uInt16** *pBuffer, **tUeiEventCallback** pEventCallback)

  Write 16 bits wide raw scans asynchronously to the stream.

- UeiDaqAPI int **UeiDaqWriteRawData32** (**SessionHandle** sessionHandle, **Int32** timeout, **Int32** numScans, **uInt32** *pBuffer)

  Write 32-bit wide raw scans to the stream.

- UeiDaqAPI int **UeiDaqWriteRawData32Async** (**SessionHandle** sessionHandle, **Int32** numScans, **uInt32** *pBuffer, **tUeiEventCallback** pEventCallback)

  Write 32 bits wide raw scans asynchronously to the stream.

- UeiDaqAPI int **UeiDaqWriteScaledData** (**SessionHandle** sessionHandle, **Int32** timeout, **Int32** numScans, **f64** *pBuffer)

  Write scaled scans to the stream.

- UeiDaqAPI int **UeiDaqWriteScaledDataAsync** (**SessionHandle** sessionHandle, **Int32** numScans, **f64** *pBuffer, **tUeiEventCallback** pEventCallback)

  Write scaled scans asynchronously to the stream.

- UeiDaqAPI int **UeiDaqWriteMessage** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numBytes, void *pBuffer, **Int32** *numBytesWritten)

  Write messages to the stream.

- UeiDaqAPI int **UeiDaqWriteMessageAsync** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numBytes, void *pBuffer, **tUeiEventCallback** pEventCallback)

  Write messages asynchronously to the stream.

- UeiDaqAPI int **UeiDaqWriteMessageInt16** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numWords, **Int16** *pBuffer, **Int32** *numWordsWritten)

Write 16-bit word messages to the stream.

- UeiDaqAPI int **UeiDaqSendSerialBreak** (**SessionHandle** sessionHandle, **Int32** port, **uInt32** durationMs)

  send serial break

- UeiDaqAPI int **UeiDaqWriteCANFrame** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numFrames, **tUeiCANFrame** *pBuffer, **Int32** *numFramesWritten)

  Write CAN frames to the stream.

- UeiDaqAPI int **UeiDaqWriteCANFrameAsync** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numFrames, **tUeiCANFrame** *pBuffer, **tUeiEventCallback** pEventCallback)

  Write CAN frames asynchronously to the stream.

- UeiDaqAPI int **UeiDaqWriteARINCWords** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numWords, **tUeiARINCWord** *pBuffer, **Int32** *numWordsWritten)

  Write ARINC words to the stream.

- UeiDaqAPI int **UeiDaqWriteARINCRawWords** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numWords, **uInt32** *pBuffer, **Int32** *numWordsWritten)

  Write RAW ARINC words to the stream.

- UeiDaqAPI int **UeiDaqWriteScheduledARINCWords** (**SessionHandle** sessionHandle, **Int32** port, **Int32** firstWord, **Int32** numWords, **tUeiARINCWord** *pBuffer, **Int32** *numWordsWritten)

  Update scheduled ARINC words.

- UeiDaqAPI int **UeiDaqWriteScheduledARINCRawWords** (**SessionHandle** sessionHandle, **Int32** port, **Int32** firstWord, **Int32** numWords, **uInt32** *pBuffer, **Int32** *numWordsWritten)

  Update scheduled ARINC words.

- UeiDaqAPI int **UeiDaqWriteARINCWordsAsync** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numWords, **tUeiARINCWord** *pBuffer, **tUeiEventCallback** pEventCallback)

  Write ARINC words asynchronously to the stream.

- UeiDaqAPI int **UeiDaqEnableARINCScheduler** (**SessionHandle** sessionHandle, **Int32** port, int enable)

  Enable scheduler.

- UeiDaqAPI int **UeiDaqSetARINCTransmitPage** (**SessionHandle** sessionHandle, **Int32** port, int immediate, **uInt32** writeMask, **uInt32** txMask)

  Set the writing page/transmit page for all channels at once.

- UeiDaqAPI int **UeiDaqWriteMIL1553Frames** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numFrames, **tUeiMIL1553Frame** *pBuffer, **Int32** *numFramesWritten)

  Write MIL-1553 frames to the stream.

- UeiDaqAPI int **UeiDaqWriteMIL1553BusWriterFrames** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numFrames, **tUeiMIL1553BusWriterFrame** *pBuffer, **Int32** *numFramesWritten)

Write MIL-1553 frames to the bus.

- UeiDaqAPI int **UeiDaqWriteMIL1553RTDataFrames** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numFrames, **tUeiMIL1553RTFrame** *pBuffer, **Int32** *numFramesWritten)

  Write MIL-1553 RT data area frames to the stream.

- UeiDaqAPI int **UeiDaqWriteMIL1553RTControlFrames** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numFrames, **tUeiMIL1553RTControlFrame** *pBuffer, **Int32** *numFramesWritten)

  Write MIL-1553 RT Control information frames to the stream.

- UeiDaqAPI int **UeiDaqWriteMIL1553RTParametersFrames** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numFrames, **tUeiMIL1553RTParametersFrame** *pBuffer, **Int32** *numFramesWritten)

  Write MIL-1553 RT set parameters frames to the stream.

- UeiDaqAPI int **UeiDaqWriteMIL1553BCCBDataFrames** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numFrames, **tUeiMIL1553BCCBDataFrame** *pBuffer, **Int32** *numFramesWritten)

  Write MIL-1553 BC control block data.

- UeiDaqAPI int **UeiDaqWriteMIL1553BCSchedFrames** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numFrames, **tUeiMIL1553BCSchedFrame** *pBuffer, **Int32** *numFramesWritten)

  Write MIL-1553 BC scheduler frames.

- UeiDaqAPI int **UeiDaqWriteMIL1553BCControlFrames** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numFrames, **tUeiMIL1553BCControlFrame** *pBuffer, **Int32** *numFramesWritten)

  Write MIL-1553 BC control frames.

- UeiDaqAPI int **UeiDaqWriteMIL1553FramesAsync** (**SessionHandle** sessionHandle, **Int32** port, **Int32** bufferSize, **tUeiMIL1553Frame** *pBuffer, **tUeiEventCallback** pEventCallback)

  Write MIL-1553 frames asynchronously to the stream.

- UeiDaqAPI int **UeiDaqWriteMIL1553A708DataFrames** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numFrames, **tUeiMIL1553A708DataFrame** *pBuffer, **Int32** *numFramesWritten)

  Write ARINC 708 data frames.

- UeiDaqAPI int **UeiDaqWriteMIL1553A708ControlFrames** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numFrames, **tUeiMIL1553A708ControlFrame** *pBuffer, **Int32** *numFramesWritten)

  Write ARINC 708 control frames.

- UeiDaqAPI int **UeiDaqWriteHDLCFrame** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numBytes, void *pBuffer, **Int32** *numBytesWritten)

  Write HDLC frames to the stream.

- UeiDaqAPI int **UeiDaqWriteAOWaveform** (**SessionHandle** sessionHandle, **Int32** channel, **Int32** numWfms, **tUeiAOWaveformParameters** *pBuffer, **Int32** *numWfmsWritten)

Write waveform to the AO device.

- UeiDaqAPI int **UeiDaqWriteAOWaveformSweep** (**SessionHandle** sessionHandle, **Int32** channel, **Int32** numSwps, **tUeiAOWaveformSweepParameters** *pBuffer, **Int32** *numSwpsWritten)

  Write sweep command to the AO device.

- UeiDaqAPI int **UeiDaqWriteAOWaveformArbitraryData** (**SessionHandle** sessionHandle, **Int32** channel, **Int32** numValues, double *pBuffer, **Int32** *numValsWritten)

  Write arbitrary waveform data to the AO device.

- UeiDaqAPI int **UeiDaqWriteSSISlaveWords** (**SessionHandle** sessionHandle, int port, int grayEncoding, **Int32** numWords, **uInt32** *pBuffer, **Int32** *numWordsWritten)

  Write word(s) to the SSI slave port.

- UeiDaqAPI int **UeiDaqResetCircuitBreaker** (**SessionHandle** sessionHandle, **Int32** port, **uInt32** mask)

  Resets circuit breaker.

- UeiDaqAPI int **UeiDaqGetCircuitBreakerStatus** (**SessionHandle** sessionHandle, **Int32** port, **uInt32** *currentStatus, **uInt32** *stickyStatus)

  Get current circuit breaker status.

- UeiDaqAPI int **UeiDaqGetAllCircuitBreakerStatus** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numStatus, **uInt32** *pBuffer, **Int32** *numStatusRead)

  Get detailed status about each circuit breaker.

- UeiDaqAPI int **UeiDaqSetAOShortOpenCircuit** (**SessionHandle** sessionHandle, **uInt32** openBitmask, **uInt32** shortBitmask, **uInt32** shortDurationUs)

  Configure short/open circuit simulation.

- UeiDaqAPI int **UeiDaqWriteCSDBFrame** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numMessages, **tUeiCSDBMessage** *pBuffer, **Int32** *numMessagesWritten)

  Write CSDB messages to the stream.

- UeiDaqAPI int **UeiDaqTriggerSync1PPSDevices** (**SessionHandle** sessionHandle, **tUeiSync1PPSTriggerOperation** trigOp, int resetTimestamp)

  send trigger signal to slave devices

- UeiDaqAPI int **UeiDaqWriteMuxRelays** (**SessionHandle** sessionHandle, int numValues, **uInt32** *relayBuffer)

  Set relays individually.

- UeiDaqAPI int **UeiDaqWriteSynchroResolverAngles** (**SessionHandle** sessionHandle, int channel, int numberOfValues, **uInt32** *pDelayBuffer, **f64** *pAngleBuffer)

  Write multiple synchro/resolver angles to be generated with fixed delay to device FIFO.

- UeiDaqAPI int **UeiDaqUpdateSynchroResolverAngles** (**SessionHandle** sessionHandle, **Int32** *written, **Int32** *available)

  Update all simulated synchro/resolver channels with staged angles and delays.

- UeiDaqAPI int **UeiDaqWriteMux** (**SessionHandle** sessionHandle, int numChannels, int *channels, int *relayIndices)

  Set mux.

- UeiDaqAPI int **UeiDaqWriteMuxRaw** (**SessionHandle** sessionHandle, int numValues, **u-Int32** *muxBuffer)

  Set low-level format mux values.

- UeiDaqAPI int **UeiDaqWriteMuxDmm** (**SessionHandle** sessionHandle, **Int32** channel-Num, **Int32** relaySelect, **tUeiMuxDmmMode** dmmMode)

  Set single channel on MUX for use with DMM.

- UeiDaqAPI int **UeiDaqWriteI2CSlaveData** (**SessionHandle** sessionHandle, **Int32** port, **Int32** numElements, **uInt16** *pBuffer, **Int32** *numElementsWritten)

  Writes data to slave.

- UeiDaqAPI int **UeiDaqWriteI2CMasterCommand** (**SessionHandle** sessionHandle, **Int32** port, **tUeiI2CMasterCommand** *command)

  Write command for master to transmit.

- UeiDaqAPI int **UeiDaqFlushMessages** (**SessionHandle** sessionHandle, **Int32** port, **tUei-FlushAction** action)

  Flush I/O buffers associated with the specifed port.

- UeiDaqAPI **Int32 UeiDaqGetAvailableInputMessages** (**SessionHandle** sessionHandle, **Int32** port, **Int32** *pAvailableInputMessages)

  Get the number of available input messages.

- UeiDaqAPI **Int32 UeiDaqGetAvailableOutputSlots** (**SessionHandle** sessionHandle, **Int32** port, **Int32** *pAvailableOutputSlots)

  Get the number of available output slots.

- UeiDaqAPI int **UeiDaqGetChannelResourceName** (**ChannelHandle** channelHandle, char *resourceName, int *resourceNameLength)

  Get a channel resource name.

- UeiDaqAPI int **UeiDaqSetChannelResourceName** (**ChannelHandle** channelHandle, char *resName)

  Set resource name.

- UeiDaqAPI int **UeiDaqGetChannelAliasName** (**ChannelHandle** channelHandle, char *aliasName, int *aliasNameLength)

  Get alias name.

- UeiDaqAPI int **UeiDaqSetChannelAliasName** (**ChannelHandle** channelHandle, char *aliasName)

  Set alias name.

- UeiDaqAPI int **UeiDaqGetChannelIndex** (**ChannelHandle** channelHandle, int *index)

  Get the index.

- UeiDaqAPI int **UeiDaqSetChannelIndex** (**ChannelHandle** channelHandle, int index)
  Set the index.

- UeiDaqAPI int **UeiDaqGetAIChannelMinimum** (**ChannelHandle** channelHandle, **f64** *minimum)
  Get the minimum range value.

- UeiDaqAPI int **UeiDaqSetAIChannelMinimum** (**ChannelHandle** channelHandle, **f64** minimum)
  Set the minimum range value.

- UeiDaqAPI int **UeiDaqGetAIChannelMaximum** (**ChannelHandle** channelHandle, **f64** *maximum)
  Get the maximum range value.

- UeiDaqAPI int **UeiDaqSetAIChannelMaximum** (**ChannelHandle** channelHandle, **f64** maximum)
  Set the maximum range value.

- UeiDaqAPI int **UeiDaqGetAIChannelInputMode** (**ChannelHandle** channelHandle, **tUeiAIChannelInputMode** *mode)
  Get the input mode.

- UeiDaqAPI int **UeiDaqGetAIChannelMuxPos** (**ChannelHandle** channelHandle, **tUeiAIChannelMuxPos** *muxPos)
  Get mux position mode.

- UeiDaqAPI int **UeiDaqSetAIChannelMuxPos** (**ChannelHandle** channelHandle, **tUeiAIChannelMuxPos** muxPos)
  Set mux position mode.

- UeiDaqAPI int **UeiDaqGetAIChannelGain** (**ChannelHandle** channelHandle, **f64** *gain)
  Get the gain.

- UeiDaqAPI int **UeiDaqSetAIChannelGain** (**ChannelHandle** channelHandle, **f64** gain)
  Set the gain.

- UeiDaqAPI int **UeiDaqSetAIChannelInputMode** (**ChannelHandle** channelHandle, **tUeiAIChannelInputMode** mode)
  Set the input mode.

- UeiDaqAPI int **UeiDaqIsAIChannelOpenCircuitTestEnabled** (**ChannelHandle** channelHandle, int *enabled)
  Get open circuit detection status.

- UeiDaqAPI int **UeiDaqEnableAIChannelOpenCircuitTest** (**ChannelHandle** channelHandle, int enable)
  Enable or disable open cicuit detection.

- UeiDaqAPI int **UeiDaqIsAIChannelCircuitOpen** (**ChannelHandle** channelHandle, int *open)

Get open circuit detection result.

- UeiDaqAPI int **UeiDaqIsAIChannelBiasEnabled** (**ChannelHandle** channelHandle, int *enabled)

  Get bias resistor state.

- UeiDaqAPI int **UeiDaqEnableAIChannelBias** (**ChannelHandle** channelHandle, int enable)

  Set bias resistor state.

- UeiDaqAPI int **UeiDaqIsAIChannelAutoZeroEnabled** (**ChannelHandle** channelHandle, int *enabled)

  Get auto-zero state.

- UeiDaqAPI int **UeiDaqEnableAIChannelAutoZero** (**ChannelHandle** channelHandle, int enable)

  Set auto zero state.

- UeiDaqAPI int **UeiDaqGetAOChannelMinimum** (**ChannelHandle** channelHandle, **f64** *minimum)

  Get the minimum value.

- UeiDaqAPI int **UeiDaqSetAOChannelMinimum** (**ChannelHandle** channelHandle, **f64** minimum)

  Set the minimum value.

- UeiDaqAPI int **UeiDaqGetAOChannelMaximum** (**ChannelHandle** channelHandle, **f64** *maximum)

  Get the maximum value.

- UeiDaqAPI int **UeiDaqSetAOChannelMaximum** (**ChannelHandle** channelHandle, **f64** maximum)

  Set the maximum value.

- UeiDaqAPI int **UeiDaqIsAOChannelDefaultValueEnabled** (**ChannelHandle** channelHandle, int *pDefaultValueEnabled)

  Get the default value state.

- UeiDaqAPI int **UeiDaqEnableAOChannelDefaultValue** (**ChannelHandle** channelHandle, int enableDefaultValue)

  Set the default value state.

- UeiDaqAPI int **UeiDaqGetAOChannelDefaultValue** (**ChannelHandle** channelHandle, **f64** *pDefaultVal)

  Get the default value.

- UeiDaqAPI int **UeiDaqSetAOChannelDefaultValue** (**ChannelHandle** channelHandle, **f64** defaultVal)

  Set the default value.

- UeiDaqAPI int **UeiDaqGetAOWaveformMainDACClockSource** (**ChannelHandle** channelHandle, **tUeiAOWaveformClockSource** *pSource)

  Get the main DAC clock source.

- UeiDaqAPI int **UeiDaqSetAOWaveformMainDACClockSource** (**ChannelHandle** channelHandle, **tUeiAOWaveformClockSource** source)

  Set the main DAC clock source.

- UeiDaqAPI int **UeiDaqGetAOWaveformOffsetDACClockSource** (**ChannelHandle** channelHandle, **tUeiAOWaveformOffsetClockSource** *pSource)

  Get the offset DAC clock source.

- UeiDaqAPI int **UeiDaqSetAOWaveformOffsetDACClockSource** (**ChannelHandle** channelHandle, **tUeiAOWaveformOffsetClockSource** source)

  Set the offset DAC clock source.

- UeiDaqAPI int **UeiDaqGetAOWaveformMainDACClockSync** (**ChannelHandle** channelHandle, **tUeiAOWaveformClockSync** *pSync)

  Get the main DAC clock synchronization.

- UeiDaqAPI int **UeiDaqSetAOWaveformMainDACClockSync** (**ChannelHandle** channelHandle, **tUeiAOWaveformClockSync** sync)

  Route the main DAC synchronization.

- UeiDaqAPI int **UeiDaqGetAOWaveformMainDACTriggerSource** (**ChannelHandle** channelHandle, **tUeiAOWaveformTriggerSource** *pSource)

  Get the main DAC trigger source.

- UeiDaqAPI int **UeiDaqSetAOWaveformMainDACTriggerSource** (**ChannelHandle** channelHandle, **tUeiAOWaveformTriggerSource** source)

  Set the main DAC trigger source.

- UeiDaqAPI int **UeiDaqGetAOWaveformOffsetDACTriggerSource** (**ChannelHandle** channelHandle, **tUeiAOWaveformOffsetTriggerSource** *pSource)

  Get the offset DAC trigger source.

- UeiDaqAPI int **UeiDaqSetAOWaveformOffsetDACTriggerSource** (**ChannelHandle** channelHandle, **tUeiAOWaveformOffsetTriggerSource** source)

  Set the offset DAC trigger source.

- UeiDaqAPI int **UeiDaqGetAOProtectedDACMode** (**ChannelHandle** channelHandle, **tUeiAODACMode** *pMode)

  Get the DAC mode.

- UeiDaqAPI int **UeiDaqSetAOProtectedDACMode** (**ChannelHandle** channelHandle, **tUeiAODACMode** mode)

  Set the DAC mode.

- UeiDaqAPI int **UeiDaqGetAOProtectedADCChannel** (**ChannelHandle** channelHandle, int index, **tUeiAODiagChannel** *pAdcChannel)

  Gets diagnostic channels.

- UeiDaqAPI int **UeiDaqSetAOProtectedADCChannel** (**ChannelHandle** channelHandle, int index, **tUeiAODiagChannel** adcChannel)

  Sets diagnostic channel.

- UeiDaqAPI int **UeiDaqGetAOProtectedCircuitBreakerLowLimit** (**ChannelHandle** channelHandle, int index, double *pLowLimit)

  Get the minimum circuit breaker limits.

- UeiDaqAPI int **UeiDaqSetAOProtectedCircuitBreakerLowLimit** (**ChannelHandle** channelHandle, int index, double lowLimit)

  Set the minimum circuit breaker limits.

- UeiDaqAPI int **UeiDaqGetAOProtectedCircuitBreakerHighLimit** (**ChannelHandle** channelHandle, int index, double *pHighLimit)

  Get the maximum circuit breaker limits.

- UeiDaqAPI int **UeiDaqSetAOProtectedCircuitBreakerHighLimit** (**ChannelHandle** channelHandle, int index, double highLimit)

  Set the maximum circuit breaker limits.

- UeiDaqAPI int **UeiDaqGetAOProtectedMeasurementRate** (**ChannelHandle** channelHandle, double *pMeasurementRate)

  Get the diagnostic measurement rate.

- UeiDaqAPI int **UeiDaqSetAOProtectedMeasurementRate** (**ChannelHandle** channelHandle, double measurementRate)

  Set the diagnostic measurement rate.

- UeiDaqAPI int **UeiDaqIsAOProtectedCircuitBreakerEnabled** (**ChannelHandle** channelHandle, int index, int *pEnabled)

  Determines whether the CB is currently protecting the channel.

- UeiDaqAPI int **UeiDaqEnableAOProtectedCircuitBreaker** (**ChannelHandle** channelHandle, int index, int enable)

  Enable or Disable channel protection.

- UeiDaqAPI int **UeiDaqGetAOProtectedAutoRetry** (**ChannelHandle** channelHandle, int *pAutoRetry)

  Get the auto retry setting.

- UeiDaqAPI int **UeiDaqSetAOProtectedAutoRetry** (**ChannelHandle** channelHandle, int autoRetry)

  Set the auto retry setting.

- UeiDaqAPI int **UeiDaqGetAOProtectedAutoRetryRate** (**ChannelHandle** channelHandle, double *pAutoRetryRate)

  Get the auto retry rate.

- UeiDaqAPI int **UeiDaqSetAOProtectedAutoRetryRate** (**ChannelHandle** channelHandle, double autoRetryRate)

Set the auto retry rate.

- UeiDaqAPI int **UeiDaqGetAOProtectedOverUnderCount** (**ChannelHandle** channelHandle, **uInt32** *pOverUnderCount)

  Get the over/under count.

- UeiDaqAPI int **UeiDaqSetAOProtectedOverUnderCount** (**ChannelHandle** channelHandle, **uInt32** overUnderCount)

  Set the over/under count.

- UeiDaqAPI int **UeiDaqGetDIChannelEdgeMask** (**ChannelHandle** channelHandle, **uInt32** *mask)

  Get the edge detection mask.

- UeiDaqAPI int **UeiDaqSetDIChannelEdgeMask** (**ChannelHandle** channelHandle, **uInt32** mask, **tUeiDigitalEdge** edgeType)

  Set the edge detection mask.

- UeiDaqAPI int **UeiDaqIsDOChannelDefaultValueEnabled** (**ChannelHandle** channelHandle, int *pDefaultValueEnabled)

  Get the default value state.

- UeiDaqAPI int **UeiDaqEnableDOChannelDefaultValue** (**ChannelHandle** channelHandle, int enableDefaultValue)

  Set the default value state.

- UeiDaqAPI int **UeiDaqGetDOChannelDefaultValue** (**ChannelHandle** channelHandle, **uInt32** *pDefaultVal)

  Get the default value.

- UeiDaqAPI int **UeiDaqSetDOChannelDefaultValue** (**ChannelHandle** channelHandle, **uInt32** defaultVal)

  Set the default value.

- UeiDaqAPI int **UeiDaqGetCIChannelCounterSource** (**ChannelHandle** channelHandle, **tUeiCounterSource** *source)

  Get the counter source.

- UeiDaqAPI int **UeiDaqSetCIChannelCounterSource** (**ChannelHandle** channelHandle, **tUeiCounterSource** source)

  Set the counter source.

- UeiDaqAPI int **UeiDaqGetCIChannelCounterMode** (**ChannelHandle** channelHandle, **tUeiCounterMode** *mode)

  Get the counter mode.

- UeiDaqAPI int **UeiDaqSetCIChannelCounterMode** (**ChannelHandle** channelHandle, **tUeiCounterMode** mode)

  Set the counter mode.

- UeiDaqAPI int **UeiDaqGetCIChannelCounterGate** (**ChannelHandle** channelHandle, **tUeiCounterGate** *gate)

  Get the counter gate.

- UeiDaqAPI int **UeiDaqSetCIChannelCounterGate** (**ChannelHandle** channelHandle, **tUeiCounterGate** gate)

  Set the counter gate.

- UeiDaqAPI int **UeiDaqGetCIChannelInverted** (**ChannelHandle** channelHandle, **Int32** *inverted)

  Get the inverted setting.

- UeiDaqAPI int **UeiDaqSetCIChannelInverted** (**ChannelHandle** channelHandle, **Int32** inverted)

  Set the inverted setting.

- UeiDaqAPI int **UeiDaqGetCITimebaseDivider** (**ChannelHandle** channelHandle, **Int32** *divider)

  Get the timebase divider.

- UeiDaqAPI int **UeiDaqSetCITimebaseDivider** (**ChannelHandle** channelHandle, **Int32** divider)

  Set the timebase divider.

- UeiDaqAPI int **UeiDaqGetCIMinimumSourcePulseWidth** (**ChannelHandle** channelHandle, double *minWidth)

  Get the minimum pulse width at the counter source.

- UeiDaqAPI int **UeiDaqSetCIMinimumSourcePulseWidth** (**ChannelHandle** channelHandle, double minWidth)

  Set the minimum pulse width at the counter source.

- UeiDaqAPI int **UeiDaqGetCIMinimumGatePulseWidth** (**ChannelHandle** channelHandle, double *minWidth)

  Get the minimum pulse width at the counter gate.

- UeiDaqAPI int **UeiDaqSetCIMinimumGatePulseWidth** (**ChannelHandle** channelHandle, double minWidth)

  Set the minimum pulse width at the counter gate.

- UeiDaqAPI int **UeiDaqGetCIGateMode** (**ChannelHandle** channelHandle, **tUeiCounterGateMode** *gateMode)

  Get the gate mode.

- UeiDaqAPI int **UeiDaqSetCIGateMode** (**ChannelHandle** channelHandle, **tUeiCounterGateMode** gateMode)

  Set the gate mode.

- UeiDaqAPI int **UeiDaqGetCIPeriodCount** (**ChannelHandle** channelHandle, **Int32** *periodCount)

  Get the period count.

- UeiDaqAPI int **UeiDaqSetCIPeriodCount** (**ChannelHandle** channelHandle, **Int32** periodCount)

  Set the period count.

- UeiDaqAPI int **UeiDaqGetCOChannelCounterSource** (**ChannelHandle** channelHandle, **tUeiCounterSource** *source)

  Get the counter source.

- UeiDaqAPI int **UeiDaqSetCOChannelCounterSource** (**ChannelHandle** channelHandle, **tUeiCounterSource** source)

  Set the counter source.

- UeiDaqAPI int **UeiDaqGetCOChannelCounterMode** (**ChannelHandle** channelHandle, **tUeiCounterMode** *mode)

  Get the counter mode.

- UeiDaqAPI int **UeiDaqSetCOChannelCounterMode** (**ChannelHandle** channelHandle, **tUeiCounterMode** mode)

  Set the counter mode.

- UeiDaqAPI int **UeiDaqGetCOChannelCounterGate** (**ChannelHandle** channelHandle, **tUeiCounterGate** *gate)

  Get the counter gate.

- UeiDaqAPI int **UeiDaqSetCOChannelCounterGate** (**ChannelHandle** channelHandle, **tUeiCounterGate** gate)

  Set the counter gate.

- UeiDaqAPI int **UeiDaqGetCOChannelInverted** (**ChannelHandle** channelHandle, **Int32** *inverted)

  Get the inverted setting.

- UeiDaqAPI int **UeiDaqSetCOChannelInverted** (**ChannelHandle** channelHandle, **Int32** inverted)

  Set the inverted setting.

- UeiDaqAPI int **UeiDaqGetCOChannelTick1** (**ChannelHandle** channelHandle, **Int32** *tick1)

  Get the number of low ticks.

- UeiDaqAPI int **UeiDaqSetCOChannelTick1** (**ChannelHandle** channelHandle, **Int32** tick1)

  Set the number of low ticks.

- UeiDaqAPI int **UeiDaqGetCOChannelTick2** (**ChannelHandle** channelHandle, **Int32** *tick2)

  Get the number of high ticks.

- UeiDaqAPI int **UeiDaqSetCOChannelTick2** (**ChannelHandle** channelHandle, **Int32** tick2)

Set the number of high ticks.

- UeiDaqAPI int **UeiDaqGetCOTimebaseDivider** (**ChannelHandle** channelHandle, **Int32** *divider)

  Get the timebase divider.

- UeiDaqAPI int **UeiDaqSetCOTimebaseDivider** (**ChannelHandle** channelHandle, **Int32** divider)

  Set the timebase divider.

- UeiDaqAPI int **UeiDaqGetCOMinimumSourcePulseWidth** (**ChannelHandle** channelHandle, double *minWidth)

  Get the minimum pulse width at the counter source.

- UeiDaqAPI int **UeiDaqSetCOMinimumSourcePulseWidth** (**ChannelHandle** channelHandle, double minWidth)

  Set the minimum pulse width at the counter source.

- UeiDaqAPI int **UeiDaqGetCOMinimumGatePulseWidth** (**ChannelHandle** channelHandle, double *minWidth)

  Get the minimum pulse width at the counter gate.

- UeiDaqAPI int **UeiDaqSetCOMinimumGatePulseWidth** (**ChannelHandle** channelHandle, double minWidth)

  Set the minimum pulse width at the counter gate.

- UeiDaqAPI int **UeiDaqGetCOGateMode** (**ChannelHandle** channelHandle, **tUeiCounterGateMode** *gateMode)

  Get the gate mode.

- UeiDaqAPI int **UeiDaqSetCOGateMode** (**ChannelHandle** channelHandle, **tUeiCounterGateMode** gateMode)

  Set the gate mode.

- UeiDaqAPI int **UeiDaqGetCONumberOfPulses** (**ChannelHandle** channelHandle, **Int32** *numberOfPulses)

  Get the pulse count.

- UeiDaqAPI int **UeiDaqSetCONumberOfPulses** (**ChannelHandle** channelHandle, **Int32** numberOfPulses)

  Set the pulse count.

- UeiDaqAPI int **UeiDaqGetTCThermocoupleType** (**ChannelHandle** channelHandle, **tUeiThermocoupleType** *pTCType)

  Get the thermocouple type.

- UeiDaqAPI int **UeiDaqSetTCThermocoupleType** (**ChannelHandle** channelHandle, **tUeiThermocoupleType** tcType)

  Set the thermocouple type.

- UeiDaqAPI int **UeiDaqGetTCTemperatureScale** (**ChannelHandle** channelHandle, **tUeiTemperatureScale** *pTempScale)

  Get the temperature scale.

- UeiDaqAPI int **UeiDaqSetTCTemperatureScale** (**ChannelHandle** channelHandle, **tUeiTemperatureScale** tempScale)

  Set the temperature scale type.

- UeiDaqAPI int **UeiDaqGetTCCJCType** (**ChannelHandle** channelHandle, **tUeiColdJunctionCompensationType** *pCJCType)

  Get the cold junction compensation type.

- UeiDaqAPI int **UeiDaqSetTCCJCType** (**ChannelHandle** channelHandle, **tUeiColdJunctionCompensationType** cjcType)

  Set the cold junction compensation type.

- UeiDaqAPI int **UeiDaqGetTCCJCResource** (**ChannelHandle** channelHandle, char *cjcResource, int *cjcResourceLength)

  Get the cold junction compensation resource.

- UeiDaqAPI int **UeiDaqSetTCCJCResource** (**ChannelHandle** channelHandle, char *cjcResource)

  Set the cold junction compensation resource.

- UeiDaqAPI int **UeiDaqGetTCCJCConstant** (**ChannelHandle** channelHandle, double *pCJCConstant)

  Get the cold junction compensation constant.

- UeiDaqAPI int **UeiDaqSetTCCJCConstant** (**ChannelHandle** channelHandle, double cjcConstant)

  Set the cold junction compensation constant.

- UeiDaqAPI int **UeiDaqGetResistanceWiringScheme** (**ChannelHandle** channelHandle, **tUeiWiringScheme** *pWiring)

  Get the wiring scheme.

- UeiDaqAPI int **UeiDaqSetResistanceWiringScheme** (**ChannelHandle** channelHandle, **tUeiWiringScheme** wiring)

  Set the resistance channel wiring scheme.

- UeiDaqAPI int **UeiDaqGetResistanceExcitationVoltage** (**ChannelHandle** channelHandle, double *pVex)

  Get the excitation voltage.

- UeiDaqAPI int **UeiDaqSetResistanceExcitationVoltage** (**ChannelHandle** channelHandle, double vex)

  Set the excitation voltage.

- UeiDaqAPI int **UeiDaqGetResistanceReferenceResistorType** (**ChannelHandle** channelHandle, **tUeiReferenceResistorType** *pRefResistorType)

  Get the reference resistor type.

- UeiDaqAPI int **UeiDaqSetResistanceReferenceResistorType** (**ChannelHandle** channelHandle, **tUeiReferenceResistorType** refResistorType)

  Set the reference resistor type.

- UeiDaqAPI int **UeiDaqGetResistanceReferenceResistance** (**ChannelHandle** channelHandle, double *pRref)

  Get the reference resistance.

- UeiDaqAPI int **UeiDaqSetResistanceReferenceResistance** (**ChannelHandle** channelHandle, double rref)

  Set the reference resistance.

- UeiDaqAPI int **UeiDaqGetRTDType** (**ChannelHandle** channelHandle, **tUeiRTDType** *pRTDType)

  Get the RTD type.

- UeiDaqAPI int **UeiDaqSetRTDType** (**ChannelHandle** channelHandle, **tUeiRTDType** type)

  Set the RTD type.

- UeiDaqAPI int **UeiDaqGetRTDNominalResistance** (**ChannelHandle** channelHandle, double *pR0)

  Get the RTD nominal resistance.

- UeiDaqAPI int **UeiDaqSetRTDNominalResistance** (**ChannelHandle** channelHandle, double r0)

  Set the RTD nominal resistance.

- UeiDaqAPI int **UeiDaqGetRTDCoefficientA** (**ChannelHandle** channelHandle, double *pA)

  Get the Callendar Van-Dusen coefficient A.

- UeiDaqAPI int **UeiDaqSetRTDCoefficientA** (**ChannelHandle** channelHandle, double A)

  Set the Callendar Van-Dusen coefficient A.

- UeiDaqAPI int **UeiDaqGetRTDCoefficientB** (**ChannelHandle** channelHandle, double *pB)

  Get the Callendar Van-Dusen coefficient B.

- UeiDaqAPI int **UeiDaqSetRTDCoefficientB** (**ChannelHandle** channelHandle, double B)

  Set the Callendar Van-Dusen coefficient B.

- UeiDaqAPI int **UeiDaqGetRTDCoefficientC** (**ChannelHandle** channelHandle, double *pC)

  Get the Callendar Van-Dusen coefficient C.

- UeiDaqAPI int **UeiDaqSetRTDCoefficientC** (**ChannelHandle** channelHandle, double C)

  Set the Callendar Van-Dusen coefficient C.

- UeiDaqAPI int **UeiDaqGetRTDTemperatureScale** (**ChannelHandle** channelHandle, **tUeiTemperatureScale** *pTempScale)

  Get the RTD channel temperature scale.

- UeiDaqAPI int **UeiDaqSetRTDTemperatureScale** (**ChannelHandle** channelHandle, **tUeiTemperatureScale** tempScale)

  Set the RTD channel temperature scale type.

- UeiDaqAPI int **UeiDaqGetAIVExBridgeType** (**ChannelHandle** channelHandle, **tUeiSensorBridgeType** *pBridgeType)

  Get the bridge type.

- UeiDaqAPI int **UeiDaqSetAIVExBridgeType** (**ChannelHandle** channelHandle, **tUeiSensorBridgeType** type)

  Set the bridge type.

- UeiDaqAPI int **UeiDaqGetAIVExExcitationVoltage** (**ChannelHandle** channelHandle, double *pVex)

  Get the excitation voltage.

- UeiDaqAPI int **UeiDaqSetAIVExExcitationVoltage** (**ChannelHandle** channelHandle, double vex)

  Set the excitation voltage.

- UeiDaqAPI int **UeiDaqGetAIVExMeasuredExcitationVoltage** (**ChannelHandle** channelHandle, double *pVex)

  Get the actual excitation voltage.

- UeiDaqAPI int **UeiDaqGetAIVExExcitationFrequency** (**ChannelHandle** channelHandle, double *pFex)

  Get the excitation frequency.

- UeiDaqAPI int **UeiDaqSetAIVExExcitationFrequency** (**ChannelHandle** channelHandle, double fex)

  Set the excitation frequency.

- UeiDaqAPI int **UeiDaqGetAIVExActualExcitationFrequency** (**ChannelHandle** channelHandle, double *pFex)

  Get the actual excitation frequency.

- UeiDaqAPI int **UeiDaqGetAIVExScalingWithExcitation** (**ChannelHandle** channelHandle, int *pScaleWithExcitation)

  Get the scaling mode.

- UeiDaqAPI int **UeiDaqSetAIVExScalingWithExcitation** (**ChannelHandle** channelHandle, int scaleWithExcitation)

  Set the scaling mode.

- UeiDaqAPI int **UeiDaqIsAIVExShuntCalibrationEnabled** (**ChannelHandle** channelHandle, int *pIsShuntCalEnabled)

  Get the shunt calibration resistor status.

- UeiDaqAPI int **UeiDaqEnableAIVExShuntCalibration** (**ChannelHandle** channelHandle, int engageShuntCal)

  Control the shunt calibration resistor.

- UeiDaqAPI int **UeiDaqGetAIVExShuntLocation** (**ChannelHandle** channelHandle, **tUeiWheatstoneBridgeBranch** *pShuntedBranch)

  Get the shunt resistor location.

- UeiDaqAPI int **UeiDaqSetAIVExShuntLocation** (**ChannelHandle** channelHandle, **tUeiWheatstoneBridgeBranch** shuntedBranch)

  Set the shunt resistor location.

- UeiDaqAPI int **UeiDaqGetAIVExShuntResistance** (**ChannelHandle** channelHandle, double *pResistance)

  Get the shunt resistance.

- UeiDaqAPI int **UeiDaqSetAIVExShuntResistance** (**ChannelHandle** channelHandle, double resistance)

  Set the shunt resistance.

- UeiDaqAPI int **UeiDaqGetAIVExActualShuntResistance** (**ChannelHandle** channelHandle, double *pActualResistance)

  Get the actual shunt resistance.

- UeiDaqAPI int **UeiDaqGetAIVExGainAdjustmentFactor** (**ChannelHandle** channelHandle, double *pGaf)

  Get the gain adjustment factor.

- UeiDaqAPI int **UeiDaqSetAIVExGainAdjustmentFactor** (**ChannelHandle** channelHandle, double gaf)

  Set the gain adjustment factore.

- UeiDaqAPI int **UeiDaqGetAIVExWiringScheme** (**ChannelHandle** channelHandle, **tUeiWiringScheme** *pWiring)

  Get the wiring scheme.

- UeiDaqAPI int **UeiDaqSetAIVExWiringScheme** (**ChannelHandle** channelHandle, **tUeiWiringScheme** wiring)

  Set the wiring schme.

- UeiDaqAPI int **UeiDaqIsAIVExOffsetNullingEnabled** (**ChannelHandle** channelHandle, int *nullingEnabled)

  Get the offset nulling circuitry status.

- UeiDaqAPI int **UeiDaqEnableAIVExOffsetNulling** (**ChannelHandle** channelHandle, int enableNulling)

  Control the offset nulling circuitry.

- UeiDaqAPI int **UeiDaqGetAIVExOffsetNullingSetting** (**ChannelHandle** channelHandle, double *setting)

Get the offset nulling setting.

- UeiDaqAPI int **UeiDaqSetAIVExOffsetNullingSetting** (**ChannelHandle** channelHandle, double setting)

  Set the offset nulling setting.

- UeiDaqAPI int **UeiDaqGetAIVExBridgeCompletionSetting** (**ChannelHandle** channelHandle, double *setting)

  Get the bridge completion setting.

- UeiDaqAPI int **UeiDaqSetAIVExBridgeCompletionSetting** (**ChannelHandle** channelHandle, double setting)

  Set the offset nulling setting.

- UeiDaqAPI int **UeiDaqGetAccelCouplingType** (**ChannelHandle** channelHandle, **tUeiCoupling** *pCoupling)

  Get the coupling type.

- UeiDaqAPI int **UeiDaqSetAccelCouplingType** (**ChannelHandle** channelHandle, **tUeiCoupling** coupling)

  Set the coupling type.

- UeiDaqAPI int **UeiDaqIsAccelLowPassFilterEnabled** (**ChannelHandle** channelHandle, int *pEnabled)

  Get the low-pass filter state.

- UeiDaqAPI int **UeiDaqEnableAccelLowPassfilter** (**ChannelHandle** channelHandle, int enabled)

  Enable or Disable the low-pass filter.

- UeiDaqAPI int **UeiDaqGetAccelSensorSensitivity** (**ChannelHandle** channelHandle, double *pSensitivity)

  Get the sensor sensitivity.

- UeiDaqAPI int **UeiDaqSetAccelSensorSensitivity** (**ChannelHandle** channelHandle, double sensitivity)

  Set the sensor sensitivity.

- UeiDaqAPI int **UeiDaqGetAccelExcitationCurrent** (**ChannelHandle** channelHandle, double *pExcCurrent)

  Get the excitation current.

- UeiDaqAPI int **UeiDaqSetAccelExcitationCurrent** (**ChannelHandle** channelHandle, double excCurrent)

  Set the excitation current.

- UeiDaqAPI int **UeiDaqGetAccelLowExcitationComparator** (**ChannelHandle** channelHandle, double *pLowComparator)

  Get the excitation low comparator voltage.

- UeiDaqAPI int **UeiDaqSetAccelLowExcitationComparator** (**ChannelHandle** channelHandle, double lowComparator)

  Set the excitation low comparator voltage.

- UeiDaqAPI int **UeiDaqGetAccelHighExcitationComparator** (**ChannelHandle** channelHandle, double *pHighComparator)

  Get the excitation high comparator voltage.

- UeiDaqAPI int **UeiDaqSetAccelHighExcitationComparator** (**ChannelHandle** channelHandle, double highComparator)

  Set the excitation high comparator voltage.

- UeiDaqAPI int **UeiDaqGetAccelLowAlarmStatus** (**ChannelHandle** channelHandle, int *pLowStatus)

  Get the low alarm status.

- UeiDaqAPI int **UeiDaqGetAccelHighAlarmStatus** (**ChannelHandle** channelHandle, int *pHighStatus)

  Get the high alarm status.

- UeiDaqAPI int **UeiDaqGetLVDTWiringScheme** (**ChannelHandle** channelHandle, **tUeiLVDTWiringScheme** *pWiring)

  Get the wiring scheme.

- UeiDaqAPI int **UeiDaqSetLVDTWiringScheme** (**ChannelHandle** channelHandle, **tUeiLVDTWiringScheme** wiring)

  Set the wiring scheme.

- UeiDaqAPI int **UeiDaqIsLVDTExternalExcitationEnabled** (**ChannelHandle** channelHandle, int *pEnabled)

  Get the external excitation state.

- UeiDaqAPI int **UeiDaqEnableLVDTExternalExcitation** (**ChannelHandle** channelHandle, int enabled)

  Enable or Disable external excitation.

- UeiDaqAPI int **UeiDaqGetLVDTExcitationVoltage** (**ChannelHandle** channelHandle, double *pVex)

  Get the excitation RMS voltage.

- UeiDaqAPI int **UeiDaqSetLVDTExcitationVoltage** (**ChannelHandle** channelHandle, double vex)

  Set the excitation RMS voltage.

- UeiDaqAPI int **UeiDaqGetLVDTExcitationFrequency** (**ChannelHandle** channelHandle, double *pFex)

  Get the excitation frequency.

- UeiDaqAPI int **UeiDaqSetLVDTExcitationFrequency** (**ChannelHandle** channelHandle, double fex)

  Set the excitation frequency.

- UeiDaqAPI int **UeiDaqGetLVDTSensorSensitivity** (**ChannelHandle** channelHandle, double *pSensitivity)

  Get the sensor sensitivity.

- UeiDaqAPI int **UeiDaqSetLVDTSensorSensitivity** (**ChannelHandle** channelHandle, double sensitivity)

  Set the sensor sensitivity.

- UeiDaqAPI int **UeiDaqGetSimulatedLVDTWiringScheme** (**ChannelHandle** channelHandle, **tUeiLVDTWiringScheme** *pWiring)

  Get the wiring scheme.

- UeiDaqAPI int **UeiDaqSetSimulatedLVDTWiringScheme** (**ChannelHandle** channelHandle, **tUeiLVDTWiringScheme** wiring)

  Set the wiring scheme.

- UeiDaqAPI int **UeiDaqGetSimulatedLVDTExcitationVoltage** (**ChannelHandle** channelHandle, double *pVex)

  Get the excitation RMS voltage.

- UeiDaqAPI int **UeiDaqSetSimulatedLVDTExcitationVoltage** (**ChannelHandle** channelHandle, double vex)

  Set the excitation RMS voltage.

- UeiDaqAPI int **UeiDaqGetSimulatedLVDTExcitationFrequency** (**ChannelHandle** channelHandle, double *pFex)

  Get the excitation frequency.

- UeiDaqAPI int **UeiDaqSetSimulatedLVDTExcitationFrequency** (**ChannelHandle** channelHandle, double fex)

  Set the excitation frequency.

- UeiDaqAPI int **UeiDaqGetSimulatedLVDTSensorSensitivity** (**ChannelHandle** channelHandle, double *pSensitivity)

  Get the sensor sensitivity.

- UeiDaqAPI int **UeiDaqSetSimulatedLVDTSensorSensitivity** (**ChannelHandle** channelHandle, double sensitivity)

  Set the sensor sensitivity.

- UeiDaqAPI int **UeiDaqGetSynchroResolverMode** (**ChannelHandle** channelHandle, **tUeiSynchroResolverMode** *pMode)

  Get the type of input sensor.

- UeiDaqAPI int **UeiDaqSetSynchroResolverMode** (**ChannelHandle** channelHandle, **tUeiSynchroResolverMode** mode)

  Set the type of input sensor.

- UeiDaqAPI int **UeiDaqIsSynchroResolverExternalExcitationEnabled** (**ChannelHandle** channelHandle, int *pEnabled)

Get the external excitation state.

- UeiDaqAPI int **UeiDaqEnableSynchroResolverExternalExcitation** (**ChannelHandle** channelHandle, int enabled)

  Enable or Disable external excitation.

- UeiDaqAPI int **UeiDaqGetSynchroResolverExcitationVoltage** (**ChannelHandle** channelHandle, double *pVex)

  Get the excitation RMS voltage.

- UeiDaqAPI int **UeiDaqSetSynchroResolverExcitationVoltage** (**ChannelHandle** channelHandle, double vex)

  Set the excitation RMS voltage.

- UeiDaqAPI int **UeiDaqGetSynchroResolverExcitationFrequency** (**ChannelHandle** channelHandle, double *pFex)

  Get the excitation frequency.

- UeiDaqAPI int **UeiDaqSetSynchroResolverExcitationFrequency** (**ChannelHandle** channelHandle, double fex)

  Set the excitation frequency.

- UeiDaqAPI int **UeiDaqGetSimulatedSynchroResolverMode** (**ChannelHandle** channelHandle, **tUeiSynchroResolverMode** *pMode)

  Get the type of sensor simulated.

- UeiDaqAPI int **UeiDaqSetSimulatedSynchroResolverMode** (**ChannelHandle** channelHandle, **tUeiSynchroResolverMode** mode)

  Set the type of sensor simulated.

- UeiDaqAPI int **UeiDaqIsSimulatedSynchroResolverExternalExcitationEnabled** (**ChannelHandle** channelHandle, int *pEnabled)

  Get the external excitation state.

- UeiDaqAPI int **UeiDaqEnableSimulatedSynchroResolverExternalExcitation** (**ChannelHandle** channelHandle, int enabled)

  Enable or Disable external excitation.

- UeiDaqAPI int **UeiDaqGetSimulatedSynchroResolverExcitationVoltage** (**ChannelHandle** channelHandle, double *pVex)

  Get the excitation RMS voltage.

- UeiDaqAPI int **UeiDaqSetSimulatedSynchroResolverExcitationVoltage** (**ChannelHandle** channelHandle, double vex)

  Set the excitation RMS voltage.

- UeiDaqAPI int **UeiDaqGetSimulatedSynchroResolverExcitationFrequency** (**ChannelHandle** channelHandle, double *pFex)

  Get the excitation frequency.

- UeiDaqAPI int **UeiDaqSetSimulatedSynchroResolverExcitationFrequency** (**ChannelHandle** channelHandle, double fex)

  Set the excitation frequency.

- UeiDaqAPI int **UeiDaqGetDOProtectedUnderCurrentLimit** (**ChannelHandle** channelHandle, int line, double *pUnderCurrent)

  Get the undercurrent limit.

- UeiDaqAPI int **UeiDaqSetDOProtectedUnderCurrentLimit** (**ChannelHandle** channelHandle, int line, double underCurrent)

  Set the undercurrent limit.

- UeiDaqAPI int **UeiDaqGetDOProtectedOverCurrentLimit** (**ChannelHandle** channelHandle, int line, double *pOverCurrent)

  Get the overcurrent limit.

- UeiDaqAPI int **UeiDaqSetDOProtectedOverCurrentLimit** (**ChannelHandle** channelHandle, int line, double overCurrent)

  Set the overcurrent limit.

- UeiDaqAPI int **UeiDaqGetDOProtectedCurrentMeasurementRate** (**ChannelHandle** channelHandle, double *pMeasurementRate)

  Get the current measurement rate.

- UeiDaqAPI int **UeiDaqSetDOProtectedCurrentMeasurementRate** (**ChannelHandle** channelHandle, double measurementRate)

  Set the current measurement rate.

- UeiDaqAPI int **UeiDaqIsDOProtectedCircuitBreakerEnabled** (**ChannelHandle** channelHandle, int line, int *pEnabled)

  Determines whether the CB is currently protecting the output line.

- UeiDaqAPI int **UeiDaqEnableDOProtectedCircuitBreaker** (**ChannelHandle** channelHandle, int line, int enable)

  Enable or Disable channel protection.

- UeiDaqAPI int **UeiDaqGetDOProtectedAutoRetry** (**ChannelHandle** channelHandle, int *pAutoRetry)

  Get the auto retry setting.

- UeiDaqAPI int **UeiDaqSetDOProtectedAutoRetry** (**ChannelHandle** channelHandle, int autoRetry)

  Set the auto retry setting.

- UeiDaqAPI int **UeiDaqGetDOProtectedAutoRetryRate** (**ChannelHandle** channelHandle, double *pAutoRetryRate)

  Get the auto retry rate.

- UeiDaqAPI int **UeiDaqSetDOProtectedAutoRetryRate** (**ChannelHandle** channelHandle, double autoRetryRate)

  Set the auto retry rate.

- UeiDaqAPI int **UeiDaqGetDOProtectedOverUnderCount** (**ChannelHandle** channelHandle, **uInt32** *pOverUnderCount)

  Get the over/under count.

- UeiDaqAPI int **UeiDaqSetDOProtectedOverUnderCount** (**ChannelHandle** channelHandle, **uInt32** overUnderCount)

  Set the over/under count.

- UeiDaqAPI int **UeiDaqGetDOProtectedPWMMode** (**ChannelHandle** channelHandle, int line, **tUeiDOPWMMode** *pMode)

  Get the PWM mode.

- UeiDaqAPI int **UeiDaqSetDOProtectedPWMMode** (**ChannelHandle** channelHandle, int line, **tUeiDOPWMMode** mode)

  Set the PWM mode.

- UeiDaqAPI int **UeiDaqGetDOProtectedPWMLength** (**ChannelHandle** channelHandle, int line, **uInt32** *pLengthus)

  Get the pulse train length.

- UeiDaqAPI int **UeiDaqSetDOProtectedPWMLength** (**ChannelHandle** channelHandle, int line, **uInt32** lengthus)

  Set pulse train length.

- UeiDaqAPI int **UeiDaqGetDOProtectedPWMPeriod** (**ChannelHandle** channelHandle, int line, **uInt32** *pPeriodus)

  Get the pulse train period.

- UeiDaqAPI int **UeiDaqSetDOProtectedPWMPeriod** (**ChannelHandle** channelHandle, int line, **uInt32** periodus)

  Set pulse train period.

- UeiDaqAPI int **UeiDaqGetDOProtectedPWMDutyCycle** (**ChannelHandle** channelHandle, int line, double *pDutyCycle)

  Get the pulse train duty cycle.

- UeiDaqAPI int **UeiDaqSetDOProtectedPWMDutyCycle** (**ChannelHandle** channelHandle, int line, double dutyCycle)

  Set pulse train duty cycle.

- UeiDaqAPI int **UeiDaqGetDIIndustrialMinimumPulseWidth** (**ChannelHandle** channelHandle, int line, double *pMinWidth)

  Get the digital input filter minimum pulse width.

- UeiDaqAPI int **UeiDaqSetDIIndustrialMinimumPulseWidth** (**ChannelHandle** channelHandle, int line, double minWidth)

  Set the digital input filter minimum pulse width.

- UeiDaqAPI int **UeiDaqGetDIIndustrialLowThreshold** (**ChannelHandle** channelHandle, int line, double *pLowThreshold)

Get the low input threshold.

- UeiDaqAPI int **UeiDaqSetDIIndustrialLowThreshold** (**ChannelHandle** channelHandle, int line, double lowThreshold)

  Set the low input threshold.

- UeiDaqAPI int **UeiDaqGetDIIndustrialHighThreshold** (**ChannelHandle** channelHandle, int line, double *pHighThreshold)

  Get the high input threshold.

- UeiDaqAPI int **UeiDaqSetDIIndustrialHighThreshold** (**ChannelHandle** channelHandle, int line, double highThreshold)

  Set the high input threshold.

- UeiDaqAPI int **UeiDaqIsDIIndustrialACModeEnabled** (**ChannelHandle** channelHandle, int line, int *pAcMode)

  Get AC mode.

- UeiDaqAPI int **UeiDaqEnableDIIndustrialACMode** (**ChannelHandle** channelHandle, int line, int acMode)

  Enable or disable AC mode.

- UeiDaqAPI int **UeiDaqGetDIIndustrialMux** (**ChannelHandle** channelHandle, int line, **t-UeiDigitalInputMux** *pMux)

  Get the digital input mux configuration.

- UeiDaqAPI int **UeiDaqSetDIIndustrialMux** (**ChannelHandle** channelHandle, int line, **t-UeiDigitalInputMux** mux)

  Set the digital input mux configuration.

- UeiDaqAPI double **UeiDaqGetDIIndustrialVoltageSupply** (**ChannelHandle** channelHandle, int line, double *pVoltage)

  Get the test/pull-up voltage.

- UeiDaqAPI int **SetDIIndustrialVoltageSupply** (**ChannelHandle** channelHandle, int line, double voltage)

  Set the test/pull-up voltage.

- UeiDaqAPI int **UeiDaqGetDIIndustrialInputGain** (**ChannelHandle** channelHandle, int line, int *pInputGain)

  Get the input gain.

- UeiDaqAPI int **UeiDaqSetDIIndustrialInputGain** (**ChannelHandle** channelHandle, int line, int inputGain)

  Set the input gain.

- UeiDaqAPI int **UeiDaqGetDIIndustrialMuxDelay** (**ChannelHandle** channelHandle, int line, int *pMuxDelayUs)

  Get the mux delay.

- UeiDaqAPI int **UeiDaqSetDIIndustrialMuxDelay** (**ChannelHandle** channelHandle, int line, int muxDelayUs)

  Set the mux delay.

- UeiDaqAPI int **UeiDaqGetSerialPortMode** (**ChannelHandle** channelHandle, **tUeiSerialPortMode** *pMode)

  Get the serial port mode.

- UeiDaqAPI int **UeiDaqSetSerialPortMode** (**ChannelHandle** channelHandle, **tUeiSerialPortMode** mode)

  Set the serial port mode.

- UeiDaqAPI int **UeiDaqGetSerialPortSpeed** (**ChannelHandle** channelHandle, **tUeiSerialPortSpeed** *pBitsPerSecond)

  Get the serial port speed.

- UeiDaqAPI int **UeiDaqSetSerialPortSpeed** (**ChannelHandle** channelHandle, **tUeiSerialPortSpeed** bitsPerSecond)

  Set the serial port speed.

- UeiDaqAPI int **UeiDaqGetSerialPortCustomSpeed** (**ChannelHandle** channelHandle, **uInt32** *pBitsPerSecond)

  Get the serial port custom speed.

- UeiDaqAPI int **UeiDaqSetSerialPortCustomSpeed** (**ChannelHandle** channelHandle, **uInt32** bitsPerSecond)

  Set the serial port speed.

- UeiDaqAPI int **UeiDaqGetSerialPortDataBits** (**ChannelHandle** channelHandle, **tUeiSerialPortDataBits** *pDataBits)

  Get the serial port data bits.

- UeiDaqAPI int **UeiDaqSetSerialPortDataBits** (**ChannelHandle** channelHandle, **tUeiSerialPortDataBits** dataBits)

  Set the serial port data bits.

- UeiDaqAPI int **UeiDaqGetSerialPortParity** (**ChannelHandle** channelHandle, **tUeiSerialPortParity** *pParity)

  Get the serial port parity.

- UeiDaqAPI int **UeiDaqSetSerialPortParity** (**ChannelHandle** channelHandle, **tUeiSerialPortParity** parity)

  Set the serial port parity.

- UeiDaqAPI int **UeiDaqGetStopBits** (**ChannelHandle** channelHandle, **tUeiSerialPortStopBits** *pStopBits)

  Get the serial port stop bits.

- UeiDaqAPI int **UeiDaqSetSerialPortStopBits** (**ChannelHandle** channelHandle, **tUeiSerialPortStopBits** stopBits)

  Set the serial port stop bits.

- UeiDaqAPI int **UeiDaqGetSerialPortTermination** (**ChannelHandle** channelHandle, char *pEol, int *eolLength)

  Get the "end of line" character sequence.

- UeiDaqAPI int **UeiDaqSetSerialPortTermination** (**ChannelHandle** channelHandle, char *eol)

  Set the "end of line" character sequence.

- UeiDaqAPI int **UeiDaqIsSerialPortTxTerminationResistorEnabled** (**ChannelHandle** channelHandle, int *pEnabled)

  Get the TX termination resistor state.

- UeiDaqAPI int **UeiDaqEnableSerialPortTxTerminationResistor** (**ChannelHandle** channelHandle, int enabled)

  Enable or Disable TX termination resistor.

- UeiDaqAPI int **UeiDaqIsSerialPortRxTerminationResistorEnabled** (**ChannelHandle** channelHandle, int *pEnabled)

  Get the RX termination resistor state.

- UeiDaqAPI int **UeiDaqEnableSerialPortRxTerminationResistor** (**ChannelHandle** channelHandle, int enabled)

  Enable or Disable RX termination resistor.

- UeiDaqAPI int **UeiDaqIsSerialPortErrorReportingEnabled** (**ChannelHandle** channelHandle, int *pEnabled)

  Get the error reporting state.

- UeiDaqAPI int **UeiDaqEnableSerialPortErrorReporting** (**ChannelHandle** channelHandle, int enabled)

  Enable or disable error reproting.

- UeiDaqAPI int **UeiDaqGetSerialPortFlowControl** (**ChannelHandle** channelHandle, **tUeiSerialPortFlowControl** *pFlowControl)

  Get Flow control.

- UeiDaqAPI int **UeiDaqSetSerialPortFlowControl** (**ChannelHandle** channelHandle, **tUeiSerialPortFlowControl** flowControl)

  Set Flow control.

- UeiDaqAPI int **UeiDaqIsSerialPortHDEchoSuppressionEnabled** (**ChannelHandle** channelHandle, int *pEnabled)

  Get the half-duplex echo suppression state.

- UeiDaqAPI int **UeiDaqEnableSerialPortHDEchoSuppression** (**ChannelHandle** channelHandle, int enabled)

  Enable or disable half-duplex echo suppression.

- UeiDaqAPI int **UeiDaqIsSerialPortTxAutoDisableEnabled** (**ChannelHandle** channelHandle, int *pEnabled)

Get the TX auto-disable state.

- UeiDaqAPI int **UeiDaqEnableSerialPortTxAutoDisable** (**ChannelHandle** channelHandle, int enabled)

  Enable or disable TX auto-disable.

- UeiDaqAPI int **UeiDaqIsSerialPortOnTheFlyParityBitEnabled** (**ChannelHandle** channelHandle, int *pEnabled)

  Get the on the fly parity bit state.

- UeiDaqAPI int **UeiDaqEnableSerialPortOnTheFlyParityBit** (**ChannelHandle** channelHandle, int enabled)

  Enable or disable on the fly parity bit.

- UeiDaqAPI int **UeiDaqGetCANPortSpeed** (**ChannelHandle** channelHandle, **tUeiCANPortSpeed** *pBitsPerSecond)

  Get the CAN port speed.

- UeiDaqAPI int **UeiDaqSetCANPortSpeed** (**ChannelHandle** channelHandle, **tUeiCANPortSpeed** bitsPerSecond)

  Set the CAN port speed.

- UeiDaqAPI int **UeiDaqGetCANPortFrameFormat** (**ChannelHandle** channelHandle, **tUeiCANFrameFormat** *pFrameFormat)

  Get the frame format.

- UeiDaqAPI int **UeiDaqSetCANPortFrameFormat** (**ChannelHandle** channelHandle, **tUeiCANFrameFormat** frameFormat)

  Set the frame format.

- UeiDaqAPI int **UeiDaqGetCANPortMode** (**ChannelHandle** channelHandle, **tUeiCANPortMode** *pMode)

  Get the operation mode.

- UeiDaqAPI int **UeiDaqSetCANPortMode** (**ChannelHandle** channelHandle, **tUeiCANPortMode** mode)

  Set the operation mode.

- UeiDaqAPI int **UeiDaqGetCANPortAcceptanceMask** (**ChannelHandle** channelHandle, **uInt32** *pMask)

  Get the acceptance mask.

- UeiDaqAPI int **UeiDaqSetCANPortAcceptanceMask** (**ChannelHandle** channelHandle, **uInt32** mask)

  Set the acceptance mask.

- UeiDaqAPI int **UeiDaqGetCANPortAcceptanceCode** (**ChannelHandle** channelHandle, **uInt32** *pCode)

  Get the acceptance code.

- UeiDaqAPI int **UeiDaqSetCANPortAcceptanceCode** (**ChannelHandle** channelHandle, **uInt32** code)

  Set the acceptance code.

- UeiDaqAPI int **UeiDaqAddCANPortFilterEntry** (**ChannelHandle** channelHandle, **tUeiCANFilterEntry** entry)

  Add a filter entry.

- UeiDaqAPI int **UeiDaqGetCANPortFilterEntry** (**ChannelHandle** channelHandle, int index, **tUeiCANFilterEntry** **entry)

  Retrieve a filter entry.

- UeiDaqAPI int **UeiDaqClearCANPortFilterEntries** (**ChannelHandle** channelHandle)

  Clear filter table.

- UeiDaqAPI int **UeiDaqIsCANPortWarningAndErrorLoggingEnabled** (**ChannelHandle** channelHandle, int *pEnabled)

  Determines whether bus errors and warnings are logged.

- UeiDaqAPI int **UeiDaqEnableCANPortWarningAndErrorLogging** (**ChannelHandle** channelHandle, int enable)

  Specifies whether bus errors and warnings are logged.

- UeiDaqAPI int **UeiDaqGetCANPortTransmitErrorCounter** (**ChannelHandle** channelHandle, **uInt32** *pTxErrorCounter)

  Get the transmit error counter.

- UeiDaqAPI int **UeiDaqGetCANPortReceiveErrorCounter** (**ChannelHandle** channelHandle, **uInt32** *pRxErrorCounter)

  Get the receive error counter.

- UeiDaqAPI int **UeiDaqGetCANPortErrorCodeCaptureRegister** (**ChannelHandle** channelHandle, **uInt32** *pErrorCodeCaptureReg)

  Get the error code capture register.

- UeiDaqAPI int **UeiDaqGetCANPortArbitrationLostCaptureRegister** (**ChannelHandle** channelHandle, **uInt32** *pArbLostCaptureReg)

  Get the arbitration lost capture register.

- UeiDaqAPI int **UeiDaqGetCANPortBitTimingRegisters** (**ChannelHandle** channelHandle, **uInt16** *pBtrValues)

  Get the bit timing registers BTR0 and BTR1.

- UeiDaqAPI int **UeiDaqSetCANPortBitTimingRegisters** (**ChannelHandle** channelHandle, **uInt16** btrValues)

  Set the bit timing registers BTR0 and BTR1.

- UeiDaqAPI int **UeiDaqGetARINCInputSpeed** (**ChannelHandle** channelHandle, **tUeiARINCPortSpeed** *pBitsPerSecond)

  Get the ARINC port speed.

- UeiDaqAPI int **UeiDaqSetARINCInputSpeed** (**ChannelHandle** channelHandle, **tUeiARINCPortSpeed** bitsPerSecond)

  Set the ARINC port speed.

- UeiDaqAPI int **UeiDaqGetARINCInputParity** (**ChannelHandle** channelHandle, **tUeiARINCPortParity** *pParity)

  Get the ARINC port parity.

- UeiDaqAPI int **UeiDaqSetARINCInputParity** (**ChannelHandle** channelHandle, **tUeiARINCPortParity** parity)

  Set the ARINC port parity.

- UeiDaqAPI int **UeiDaqIsARINCInputSDIFilterEnabled** (**ChannelHandle** channelHandle, int *pSDIFilterEnabled)

  Get the SDI filter state.

- UeiDaqAPI int **UeiDaqEnableARINCInputSDIFilter** (**ChannelHandle** channelHandle, int enableSDIFilter)

  Set the SDI filter state.

- UeiDaqAPI int **UeiDaqGetARINCInputSDIFilterMask** (**ChannelHandle** channelHandle, **uInt32** *pSDIFilterMask)

  Get the SDI filter mask.

- UeiDaqAPI int **UeiDaqSetARINCInputSDIFilterMask** (**ChannelHandle** channelHandle, **uInt32** SDIFilterMask)

  Set the SDI filter mask.

- UeiDaqAPI int **UeiDaqIsARINCInputTimestampingEnabled** (**ChannelHandle** channelHandle, int *pTimestampingEnabled)

  Get the timestamping state.

- UeiDaqAPI int **UeiDaqEnableARINCInputTimestamping** (**ChannelHandle** channelHandle, int enableTimestamping)

  Set the timestamping state.

- UeiDaqAPI int **UeiDaqIsARINCInputSlowSlewRateEnabled** (**ChannelHandle** channelHandle, int *pSlowSlewRateEnabled)

  Get the slow slew rate state.

- UeiDaqAPI int **UeiDaqEnableARINCInputSlowSlewRate** (**ChannelHandle** channelHandle, int enableSlowSlewRate)

  Set the slow slew state.

- UeiDaqAPI int **UeiDaqAddARINCInputFilterEntry** (**ChannelHandle** channelHandle, **tUeiARINCFilterEntry** entry)

  Add a filter entry.

- UeiDaqAPI int **UeiDaqGetARINCInputFilterEntry** (**ChannelHandle** channelHandle, int index, **tUeiARINCFilterEntry** **entry)

  Retrieve a filter entry.

- UeiDaqAPI int **UeiDaqClearARINCInputFilterEntries** (**ChannelHandle** channelHandle)
  Clear filter table.

- UeiDaqAPI int **UeiDaqIsARINCInputLabelFilterEnabled** (**ChannelHandle** channelHandle, int *pLabelFilterEnabled)
  Get the label filter state.

- UeiDaqAPI int **UeiDaqEnableARINCInputLabelFilter** (**ChannelHandle** channelHandle, int enableLabelFilter)
  Set the label filter state.

- UeiDaqAPI int **UeiDaqGetARINCOutputSpeed** (**ChannelHandle** channelHandle, **tUeiARINCPortSpeed** *pBitsPerSecond)
  Get the ARINC port speed.

- UeiDaqAPI int **UeiDaqSetARINCOutputSpeed** (**ChannelHandle** channelHandle, **tUeiARINCPortSpeed** bitsPerSecond)
  Set the ARINC port speed.

- UeiDaqAPI int **UeiDaqGetARINCOutputParity** (**ChannelHandle** channelHandle, **tUeiARINCPortParity** *pParity)
  Get the ARINC port parity.

- UeiDaqAPI int **UeiDaqSetARINCOutputParity** (**ChannelHandle** channelHandle, **tUeiARINCPortParity** parity)
  Set the ARINC port parity.

- UeiDaqAPI int **UeiDaqIsARINCOutputLoopbackEnabled** (**ChannelHandle** channelHandle, int *pLoopbackEnabled)
  Get the loopback state.

- UeiDaqAPI int **UeiDaqEnableARINCOutputLoopback** (**ChannelHandle** channelHandle, int enableLoopback)
  Set the loopback state.

- UeiDaqAPI int **UeiDaqIsARINCOutputSlowSlewRateEnabled** (**ChannelHandle** channelHandle, int *pSlowSlewRateEnabled)
  Get the slow slew rate state.

- UeiDaqAPI int **UeiDaqEnableARINCOutputSlowSlewRate** (**ChannelHandle** channelHandle, int enableSlowSlewRate)
  Set the slow slew state.

- UeiDaqAPI int **UeiDaqAddARINCOutputSchedulerEntry** (**ChannelHandle** channelHandle, **tUeiARINCSchedulerEntry** entry)
  Add a scheduler entry.

- UeiDaqAPI int **UeiDaqGetARINCOutputSchedulerEntry** (**ChannelHandle** channelHandle, int index, **tUeiARINCSchedulerEntry** **entry)
  Retrieve a scheduler entry.

- UeiDaqAPI int **UeiDaqClearARINCOutputSchedulerEntries** (**ChannelHandle** channelHandle)

  Clear scheduler table.

- UeiDaqAPI int **UeiDaqIsARINCOutputSchedulerEnabled** (**ChannelHandle** channelHandle, int *pSchedulerEnabled)

  Get the scheduler state.

- UeiDaqAPI int **UeiDaqEnableARINCOutputScheduler** (**ChannelHandle** channelHandle, int enableScheduler)

  Set the scheduler state.

- UeiDaqAPI int **UeiDaqGetARINCOutputSchedulerType** (**ChannelHandle** channelHandle, **tUeiARINCSchedulerType** *type)

  Get the scheduler type.

- UeiDaqAPI int **UeiDaqSetARINCOutputSchedulerType** (**ChannelHandle** channelHandle, **tUeiARINCSchedulerType** type)

  Set the scheduler type.

- UeiDaqAPI int **UeiDaqGetARINCOutputSchedulerRate** (**ChannelHandle** channelHandle, double *rate)

  Get the scheduler rate.

- UeiDaqAPI int **UeiDaqSetARINCOutputSchedulerRate** (**ChannelHandle** channelHandle, double rate)

  Set the scheduler rate.

- UeiDaqAPI int **UeiDaqGetARINCOutputFIFORate** (**ChannelHandle** channelHandle, double *rate)

  Get the TX FIFO rate.

- UeiDaqAPI int **UeiDaqSetARINCOutputFIFORate** (**ChannelHandle** channelHandle, double rate)

  Set the TX FIFO rate.

- UeiDaqAPI int **UeiDaqAddARINCOutputMinorFrameEntry** (**ChannelHandle** channelHandle, tUeiARINCMinorFrameEntry entry)

  Add a minor frame entry.

- UeiDaqAPI int **UeiDaqGetARINCOutputMinorFrameEntry** (**ChannelHandle** channelHandle, int index, tUeiARINCMinorFrameEntry **entry)

  Retrieve a minor frame entry.

- UeiDaqAPI int **UeiDaqClearARINCOutputMinorFrameEntries** (**ChannelHandle** channelHandle)

  Clear major frame.

- UeiDaqAPI int **UeiDaqGetMIL1553Coupling** (**ChannelHandle** channelHandle, **tUeiMIL1553PortCoupling** *pCoupling)

Get the 1553 port coupling.

- UeiDaqAPI int **UeiDaqSetMIL1553Coupling** (**ChannelHandle** channelHandle, **tUeiMIL1553PortCoupling** coupling)

  Set the 1553 port coupling.

- UeiDaqAPI int **UeiDaqGetMIL1553PortMode** (**ChannelHandle** channelHandle, **tUeiMIL1553PortOpMode** *pPortMode)

  Get the 1553 port mode of operation.

- UeiDaqAPI int **UeiDaqSetMIL1553PortMode** (**ChannelHandle** channelHandle, **tUeiMIL1553PortOpMode** portMode)

  Set the 1553 port mode of operation.

- UeiDaqAPI int **UeiDaqGetMIL1553TxBus** (**ChannelHandle** channelHandle, **tUeiMIL1553PortActiveBus** *pPortBus)

  Get the 1553 active transmission bus.

- UeiDaqAPI int **UeiDaqSetMIL1553TxBus** (**ChannelHandle** channelHandle, **tUeiMIL1553PortActiveBus** portBus)

  Set the 1553 active transmission bus.

- UeiDaqAPI int **UeiDaqGetMIL1553RxBus** (**ChannelHandle** channelHandle, **tUeiMIL1553PortActiveBus** *pPortBus)

  Get the 1553 active reception bus.

- UeiDaqAPI int **UeiDaqSetMIL1553RxBus** (**ChannelHandle** channelHandle, **tUeiMIL1553PortActiveBus** portBus)

  Set the 1553 active reception bus.

- UeiDaqAPI int **UeiDaqIsMIL1553TimestampingEnabled** (**ChannelHandle** channelHandle, int *pTimestampingEnabled)

  Get the timestamping state.

- UeiDaqAPI int **UeiDaqEnableMIL1553Timestamping** (**ChannelHandle** channelHandle, int enableTimestamping)

  Set the timestamping state.

- UeiDaqAPI int **UeiDaqAddMIL1553FilterEntry** (**ChannelHandle** channelHandle, **tUeiMIL1553FilterEntry** entry)

  Add a filter entry.

- UeiDaqAPI int **UeiDaqGetMIL1553FilterEntry** (**ChannelHandle** channelHandle, int index, **tUeiMIL1553FilterEntry** **entry)

  Retrieve a filter entry.

- UeiDaqAPI int **UeiDaqClearMIL1553FilterEntries** (**ChannelHandle** channelHandle)

  Clear filter table.

- UeiDaqAPI int **UeiDaqIsMIL1553FilterEnabled** (**ChannelHandle** channelHandle, int *pFilterEnabled)

Get the input filter state.

- UeiDaqAPI int **UeiDaqEnableMIL1553Filter** (**ChannelHandle** channelHandle, int enableFilter)

  Set the input filter state.

- UeiDaqAPI int **UeiDaqAddMIL1553SchedulerEntry** (**ChannelHandle** channelHandle, **tUeiMIL1553SchedulerEntry** entry)

  Add a scheduler entry.

- UeiDaqAPI int **UeiDaqGetMIL1553SchedulerEntry** (**ChannelHandle** channelHandle, int index, **tUeiMIL1553SchedulerEntry** **entry)

  Retrieve a scheduler entry.

- UeiDaqAPI int **UeiDaqClearMIL1553SchedulerEntries** (**ChannelHandle** channelHandle)

  Clear scheduler table.

- UeiDaqAPI int **UeiDaqIsMIL1553SchedulerEnabled** (**ChannelHandle** channelHandle, int *pSchedulerEnabled)

  Get the scheduler state.

- UeiDaqAPI int **UeiDaqEnableMIL1553Scheduler** (**ChannelHandle** channelHandle, int enableScheduler)

  Set the scheduler state.

- UeiDaqAPI int **UeiDaqGetIRIGTimeKeeper1PPSSource** (**ChannelHandle** channelHandle, **tUeiIRIGTimeKeeper1PPSSource** *source)

  Get the time keeper source.

- UeiDaqAPI int **UeiDaqSetIRIGTimeKeeper1PPSSource** (**ChannelHandle** channelHandle, **tUeiIRIGTimeKeeper1PPSSource** source)

  Set the time keeper source.

- UeiDaqAPI int **UeiDaqIsIRIGTimeKeeperAutoFollowEnabled** (**ChannelHandle** channelHandle, int *autoFollowEnabled)

  Get the auto-follow state.

- UeiDaqAPI int **UeiDaqEnableIRIGTimeKeeperAutoFollow** (**ChannelHandle** channelHandle, int enableAutoFollow)

  Set the auto-follow state.

- UeiDaqAPI int **UeiDaqIsIRIGTimeKeeperNominalValueEnabled** (**ChannelHandle** channelHandle, int *nominalValueEnabled)

  Get the nominal value state.

- UeiDaqAPI int **UeiDaqEnableIRIGTimeKeeperNominalValue** (**ChannelHandle** channelHandle, int enableNominalValue)

  Set the nominal value state.

- UeiDaqAPI int **UeiDaqIsIRIGTimeKeeperSubPPSEnabled** (**ChannelHandle** channelHandle, int *subPPSEnabled)

  Get the sub PPS state.

- UeiDaqAPI int **UeiDaqEnableIRIGTimeKeeperSubPPS** (**ChannelHandle** channelHandle, int enableSubPPS)

  Set the nominal value state.

- UeiDaqAPI int **UeiDaqIsIRIGTimeKeeperSBSEnabled** (**ChannelHandle** channelHandle, int *SBSenabled)

  Get the SBS state.

- UeiDaqAPI int **UeiDaqEnableIRIGTimeKeeperSBS** (**ChannelHandle** channelHandle, int enableSBS)

  Set the SBS state.

- UeiDaqAPI int **UeiDaqIsIRIGTimeKeeperInvalidSecondEnabled** (**ChannelHandle** channelHandle, int *invalidSecondEnabled)

  Get the invalid second state.

- UeiDaqAPI int **UeiDaqEnableIRIGTimeKeeperInvalidSecond** (**ChannelHandle** channelHandle, int enableInvalidSecond)

  Set the invalid second state.

- UeiDaqAPI int **UeiDaqIsIRIGTimeKeeperInvalidMinuteEnabled** (**ChannelHandle** channelHandle, int *invalidMinuteEnabled)

  Get the invalid minute state.

- UeiDaqAPI int **UeiDaqEnableIRIGTimeKeeperInvalidMinute** (**ChannelHandle** channelHandle, int enableInvalidMinute)

  Set the invalid minute state.

- UeiDaqAPI int **UeiDaqIsIRIGTimeKeeperInvalidDayEnabled** (**ChannelHandle** channelHandle, int *invalidDayEnabled)

  Get the invalid day state.

- UeiDaqAPI int **UeiDaqEnableIRIGTimeKeeperInvalidDay** (**ChannelHandle** channelHandle, int enableInvalidDay)

  Set the invalid day state.

- UeiDaqAPI int **UeiDaqGetIRIGInputTimeDecoderInput** (**ChannelHandle** channelHandle, **tUeiIRIGDecoderInputType** *input)

  Get the time decoder input type.

- UeiDaqAPI int **UeiDaqSetIRIGInputTimeDecoderInput** (**ChannelHandle** channelHandle, **tUeiIRIGDecoderInputType** input)

  Set the time decoder input type.

- UeiDaqAPI int **UeiDaqGetIRIGInputTimeCodeFormat** (**ChannelHandle** channelHandle, **tUeiIRIGTimeCodeFormat** *format)

  Get the time code format.

- UeiDaqAPI int **UeiDaqSetIRIGInputTimeCodeFormat** (**ChannelHandle** channelHandle, **tUeiIRIGTimeCodeFormat** format)

  Set the time code format.

- UeiDaqAPI int **UeiDaqGetVRMode** (**ChannelHandle** channelHandle, **tUeiVRMode** *mode)

  Get the current VR channel input mode.

- UeiDaqAPI int **UeiDaqSetVRMode** (**ChannelHandle** channelHandle, **tUeiVRMode** mode)

  Set the VR channel input mode.

- UeiDaqAPI int **UeiDaqGetVRZCMode** (**ChannelHandle** channelHandle, **tUeiVRZCMode** *mode)

  Get the current zero-crossing mode.

- UeiDaqAPI int **UeiDaqSetVRZCMode** (**ChannelHandle** channelHandle, **tUeiVRZCMode** mode)

  Set the zero-crossing mode.

- UeiDaqAPI int **UeiDaqGetVRAPTMode** (**ChannelHandle** channelHandle, **tUeiVRAPTMode** *mode)

  Get the current adaptive peak threshold mode.

- UeiDaqAPI int **UeiDaqSetVRAPTMode** (**ChannelHandle** channelHandle, **tUeiVRAPTMode** mode)

  Set the adaptive peak threshold mode.

- UeiDaqAPI int **UeiDaqGetVRADCRate** (**ChannelHandle** channelHandle, double *rate)

  Get the ADC measurement rate.

- UeiDaqAPI int **UeiDaqSetVRADCRate** (**ChannelHandle** channelHandle, double rate)

  Set the ADC rate.

- UeiDaqAPI int **UeiDaqGetVRADCMovingAverage** (**ChannelHandle** channelHandle, int *mvAvrg)

  Get the ADC moving average.

- UeiDaqAPI int **UeiDaqSetVRADCMovingAverage** (**ChannelHandle** channelHandle, int mvAvrg)

  Set the ADC moving average.

- UeiDaqAPI int **UeiDaqGetVRAPTThresholdDivider** (**ChannelHandle** channelHandle, int *divider)

  Get the APT threshold divider.

- UeiDaqAPI int **UeiDaqSetVRAPTThresholdDivider** (**ChannelHandle** channelHandle, int divider)

  Set the APT threshold divider.

- UeiDaqAPI int **UeiDaqGetVRAPTThreshold** (**ChannelHandle** channelHandle, double *threshold)

  Get the APT threshold.

- UeiDaqAPI int **UeiDaqSetVRAPTThreshold** (**ChannelHandle** channelHandle, double threshold)

  Set the APT threshold.

- UeiDaqAPI int **UeiDaqGetVRZCLevel** (**ChannelHandle** channelHandle, double *level)

  Get the ZC level.

- UeiDaqAPI int **UeiDaqSetVRZCLevel** (**ChannelHandle** channelHandle, double level)

  Set the ZC level.

- UeiDaqAPI int **UeiDaqGetVRNumberOfTeeth** (**ChannelHandle** channelHandle, int *numTeeth)

  Get the number of teeth on the encoder wheel.

- UeiDaqAPI int **UeiDaqSetVRNumberOfTeeth** (**ChannelHandle** channelHandle, int numTeeth)

  Set the number of teeth on the encoder wheel.

- UeiDaqAPI int **UeiDaqIsMuxBreakBeforeMakeEnabled** (**ChannelHandle** channelHandle, int *enabled)

  Get port break before make setting.

- UeiDaqAPI int **UeiDaqEnableMuxBreakBeforeMake** (**ChannelHandle** channelHandle, int enable)

  Set port break before make setting.

- UeiDaqAPI int **UeiDaqIsMuxSyncInputEnabled** (**ChannelHandle** channelHandle, int *enabled)

  Get port synchronization input setting.

- UeiDaqAPI int **UeiDaqEnableMuxSyncInput** (**ChannelHandle** channelHandle, int enable)

  Set port synchronization input setting.

- UeiDaqAPI int **UeiDaqIsMuxSyncInputEdgeModeEnabled** (**ChannelHandle** channelHandle, int *enabled)

  Get port synchronization input edge/level setting.

- UeiDaqAPI int **UeiDaqEnableMuxSyncInputEdgeMode** (**ChannelHandle** channelHandle, int enable)

  Set port synchronization input edge/level setting.

- UeiDaqAPI int **UeiDaqGetMuxSyncInputEdgePolarity** (**ChannelHandle** channelHandle, **tUeiDigitalEdge** *polarity)

  Get port synchronization input edge/level polarity.

- UeiDaqAPI int **UeiDaqSetMuxSyncInputEdgePolarity** (**ChannelHandle** channelHandle, **tUeiDigitalEdge** polarity)

  Set port synchronization input edge/level polarity.

- UeiDaqAPI int **UeiDaqGetMuxSyncOutputMode** (**ChannelHandle** channelHandle, **tUeiMuxSyncOutputMode** *mode)

  Get port synchronization output mode.

- UeiDaqAPI int **UeiDaqSetMuxSyncOutputMode** (**ChannelHandle** channelHandle, **tUeiMuxSyncOutputMode** mode)

  Set port synchronization output mode.

- UeiDaqAPI int **UeiDaqGetMuxSyncOutputPulseWidth** (**ChannelHandle** channelHandle, int *width)

  Get port synchronization output pulse width.

- UeiDaqAPI int **UeiDaqSetMuxSyncOutputPulseWidth** (**ChannelHandle** channelHandle, int width)

  Set port synchronization output pulse width.

- UeiDaqAPI int **UeiDaqGetMuxOnDelay** (**ChannelHandle** channelHandle, int *delay)

  Get port on delay.

- UeiDaqAPI int **UeiDaqSetMuxOnDelay** (**ChannelHandle** channelHandle, int delay)

  Set port on delay.

- UeiDaqAPI int **UeiDaqGetMuxOffDelay** (**ChannelHandle** channelHandle, int *delay)

  Get port off delay.

- UeiDaqAPI int **UeiDaqSetMuxOffDelay** (**ChannelHandle** channelHandle, int delay)

  Set port on delay.

- UeiDaqAPI int **UeiDaqGetI2CSpeed** (**ChannelHandle** channelHandle, **tUeiI2CPortSpeed** *bitRate)

  Get the I2C bus bit rate.

- UeiDaqAPI int **UeiDaqSetI2CSpeed** (**ChannelHandle** channelHandle, **tUeiI2CPortSpeed** bitRate)

  Set the I2C bus bit rate.

- UeiDaqAPI int **UeiDaqGetI2CCustomSpeed** (**ChannelHandle** channelHandle, float *bitRate)

  Get the I2C bus custom bit rate.

- UeiDaqAPI int **UeiDaqSetI2CCustomSpeed** (**ChannelHandle** channelHandle, float bitRate)

  Set the I2C bus custom bit rate.

- UeiDaqAPI int **UeiDaqGetI2CTTLLevel** (**ChannelHandle** channelHandle, **tUeiI2CTTLLevel** *ttlLevel)

  Get the I2C bus TTL level.

- UeiDaqAPI int **UeiDaqSetI2CTTLLevel** (**ChannelHandle** channelHandle, **tUeiI2CTTLLevel** ttlLevel)

  Set the I2C bus TTL level.

- UeiDaqAPI int **UeiDaqIsI2CMasterSecureShellEnabled** (**ChannelHandle** channelHandle, int *enabled)

  Get the I2C secure shell setting.

- UeiDaqAPI int **UeiDaqEnableI2CMasterSecureShell** (**ChannelHandle** channelHandle, int enable)

  Set the I2C secure shell setting.

- UeiDaqAPI int **UeiDaqIsI2CMultiMasterEnabled** (**ChannelHandle** channelHandle, int *enabled)

  Get the I2C multi-master setting.

- UeiDaqAPI int **UeiDaqEnableI2CMultiMaster** (**ChannelHandle** channelHandle, int enable)

  Set the I2C multi-master setting.

- UeiDaqAPI int **UeiDaqIsI2CMasterTerminationResistorEnabled** (**ChannelHandle** channelHandle, int *enabled)

  Get the I2C termination resistor setting.

- UeiDaqAPI int **UeiDaqEnableI2CMasterTerminationResistor** (**ChannelHandle** channelHandle, int enable)

  Set the I2C termination resistor setting.

- UeiDaqAPI int **UeiDaqGetI2CMasterLoopbackMode** (**ChannelHandle** channelHandle, **tUeiI2CLoopback** *mode)

  Get the I2C loopback mode.

- UeiDaqAPI int **UeiDaqSetI2CMasterLoopbackMode** (**ChannelHandle** channelHandle, **tUeiI2CLoopback** mode)

  Set the I2C loopback mode.

- UeiDaqAPI int **UeiDaqGetI2CMasterByteToByteDelay** (**ChannelHandle** channelHandle, **uInt16** *delayUs)

  Get the delay in microseconds that the master will delay between bytes.

- UeiDaqAPI int **UeiDaqSetI2CMasterByteToByteDelay** (**ChannelHandle** channelHandle, **uInt16** delayUs)

  Set the delay in microseconds that the master will delay between bytes.

- UeiDaqAPI int **UeiDaqGetI2CMasterMaxClockStretchingDelay** (**ChannelHandle** channelHandle, **uInt16** *delayUs)

  Get the time in microseconds that the master will allow a slave to delay the clock.

- UeiDaqAPI int **UeiDaqSetI2CMasterMaxClockStretchingDelay** (**ChannelHandle** channelHandle, **uInt16** delayUs)

Set the delay in microseconds that the master will allow a slave to delay the clock.

- UeiDaqAPI int **UeiDaqGetI2CSlaveAddressWidth** (**ChannelHandle** channelHandle, **tUeiI2CSlaveAddressWidth** *width)

  Get the I2C slave address width.

- UeiDaqAPI int **UeiDaqSetI2CSlaveAddressWidth** (**ChannelHandle** channelHandle, **tUeiI2CSlaveAddressWidth** width)

  Set the I2C slave address width.

- UeiDaqAPI int **UeiDaqGetI2CSlaveAddress** (**ChannelHandle** channelHandle, int *address)

  Get the I2C slave address.

- UeiDaqAPI int **UeiDaqSetI2CSlaveAddress** (**ChannelHandle** channelHandle, int address)

  Set the I2C slave address.

- UeiDaqAPI int **UeiDaqIsI2CSlaveBusMonitorEnabled** (**ChannelHandle** channelHandle, int *enabled)

  Get the I2C bus monitor setting.

- UeiDaqAPI int **UeiDaqEnableI2CSlaveBusMonitor** (**ChannelHandle** channelHandle, int enable)

  Set the I2C bus monitor setting.

- UeiDaqAPI int **UeiDaqIsI2CSlaveBusMonitorAckEnabled** (**ChannelHandle** channelHandle, int *enabled)

  Get the I2C bus monitor ACK setting.

- UeiDaqAPI int **UeiDaqEnableI2CSlaveBusMonitorAck** (**ChannelHandle** channelHandle, int enable)

  Set the I2C bus monitor ACK setting.

- UeiDaqAPI int **UeiDaqGetI2CSlaveTransmitDataMode** (**ChannelHandle** channelHandle, **tUeiI2CSlaveDataMode** *dataMode)

  Get slave transmit data mode.

- UeiDaqAPI int **UeiDaqSetI2CSlaveTransmitDataMode** (**ChannelHandle** channelHandle, **tUeiI2CSlaveDataMode** dataMode)

  Set slave transmit data mode.

- UeiDaqAPI int **UeiDaqGetI2CSlaveRegisterData** (**ChannelHandle** channelHandle, int byteIndex, **uInt8** *data)

  Get data transmitted when in Register mode or padding data when in FIFO mode.

- UeiDaqAPI int **UeiDaqSetI2CSlaveRegisterData** (**ChannelHandle** channelHandle, int byteIndex, **uInt8** data)

  Set the I2C slave padding data.

- UeiDaqAPI int **UeiDaqGetI2CSlaveRegisterDataSize** (**ChannelHandle** channelHandle, int *size)

  Get the number of bytes used in slave data when in Register data mode.

- UeiDaqAPI int **UeiDaqSetI2CSlaveRegisterDataSize** (**ChannelHandle** channelHandle, int size)

  Set the number of bytes used in slave data when in Register data mode.

- UeiDaqAPI int **UeiDaqGetI2CSlaveMaxWordsPerAck** (**ChannelHandle** channelHandle, int *maxWords)

  Get maximum number of words slave will receive per transmission.

- UeiDaqAPI int **UeiDaqSetI2CSlaveMaxWordsPerAck** (**ChannelHandle** channelHandle, int maxWords)

  Set maximum number of words slave will receive per transmission.

- UeiDaqAPI int **UeiDaqGetI2CSlaveClockStretchingDelay** (**ChannelHandle** channelHandle, **uInt16** *clocks)

  Get the number of clocks the slave will delay an ACK.

- UeiDaqAPI int **UeiDaqSetI2CSlaveClockStretchingDelay** (**ChannelHandle** channelHandle, **uInt16** clocks)

  Get the number of clocks the slave will delay an ACK.

- UeiDaqAPI int **UeiDaqIsI2CSlaveAddressClockStretchingEnabled** (**ChannelHandle** channelHandle, int *enabled)

  Query the status of clock stretching during address cycle.

- UeiDaqAPI int **UeiDaqEnableI2CSlaveAddressClockStretching** (**ChannelHandle** channelHandle, int enable)

  Enable clock stretching during address cycle.

- UeiDaqAPI int **UeiDaqIsI2CSlaveTransmitClockStretchingEnabled** (**ChannelHandle** channelHandle, int *enabled)

  Query the status of clock stretching during transmit cycle.

- UeiDaqAPI int **UeiDaqEnableI2CSlaveTransmitClockStretching** (**ChannelHandle** channelHandle, int enable)

  Enable clock stretching during transmit cycle.

- UeiDaqAPI int **UeiDaqIsI2CSlaveReceiveClockStretchingEnabled** (**ChannelHandle** channelHandle, int *enabled)

  Query the status of clock stretching during receive cycle.

- UeiDaqAPI int **UeiDaqEnableI2CSlaveReceiveClockStretching** (**ChannelHandle** channelHandle, int enable)

  Enable clock stretching during receive cycle.

- UeiDaqAPI int **UeiDaqGetDMMFIRCutoff** (**ChannelHandle** channelHandle, **tUeiDMMFIRCutoff** *frequency)

  Get the FIR cutoff frequency.

- UeiDaqAPI int **UeiDaqSetDMMFIRCutoff** (**ChannelHandle** channelHandle, **tUeiDMMFIRCutoff** frequency)

  Set the FIR cutoff frequency.

- UeiDaqAPI int **UeiDaqGetDataStreamNumberOfScans** (**DataStreamHandle** dataStreamHandle, **Int32** *numScans)

  Get the FIR decimation rateSet the FIR decimation rateGet the number of scans to read at a time.

- UeiDaqAPI int **UeiDaqSetDataStreamNumberOfScans** (**DataStreamHandle** dataStreamHandle, **Int32** numScans)

  Set the number of scans to read at a time.

- UeiDaqAPI int **UeiDaqGetDataStreamScanSize** (**DataStreamHandle** dataStreamHandle, **Int32** *scanSize)

  Get the size of a scan.

- UeiDaqAPI int **UeiDaqSetDataStreamScanSize** (**DataStreamHandle** dataStreamHandle, **Int32** scanSize)

  Set the size of a scan.

- UeiDaqAPI int **UeiDaqGetDataStreamSampleSize** (**DataStreamHandle** dataStreamHandle, **Int32** *sampleSize)

  Get the size of a raw sample.

- UeiDaqAPI int **UeiDaqGetDataStreamNumberOfFrames** (**DataStreamHandle** dataStreamHandle, **Int32** *numFrames)

  Get the number of frames.

- UeiDaqAPI int **UeiDaqSetDataStreamNumberOfFrames** (**DataStreamHandle** dataStreamHandle, **Int32** numFrames)

  Set the number of frames.

- UeiDaqAPI int **UeiDaqGetDataStreamRegenerate** (**DataStreamHandle** dataStreamHandle, **Int32** *regenerate)

  Get the regenerate setting.

- UeiDaqAPI int **UeiDaqSetDataStreamRegenerate** (**DataStreamHandle** dataStreamHandle, **Int32** regenerate)

  Set the regenerate setting.

- UeiDaqAPI int **UeiDaqGetDataStreamBurst** (**DataStreamHandle** dataStreamHandle, **Int32** *pBurst)

  Get the burst setting.

- UeiDaqAPI int **UeiDaqSetDataStreamBurst** (**DataStreamHandle** dataStreamHandle, **Int32** burst)

  Set the burst setting.

- UeiDaqAPI int **UeiDaqGetDataStreamOverUnderRun** (**DataStreamHandle** dataStreamHandle, **Int32** *overunderrun)

Get the Overrun/underrun setting.

- UeiDaqAPI int **UeiDaqSetDataStreamOverUnderRun** (**DataStreamHandle** dataStreamHandle, **Int32** overunderrun)

  Set the Overrun/underrun setting.

- UeiDaqAPI int **UeiDaqIsDataStreamBlockingEnabled** (**DataStreamHandle** dataStreamHandle, **Int32** *enabled)

  Get the value of the blocking setting.

- UeiDaqAPI int **UeiDaqEnableDataStreamBlocking** (**DataStreamHandle** dataStreamHandle, **Int32** enable)

  Set the value of the blocking setting.

- UeiDaqAPI int **UeiDaqGetDataStreamCurrentScan** (**DataStreamHandle** dataStreamHandle, **Int32** *currentScan)

  Get the current scan position.

- UeiDaqAPI int **UeiDaqGetDataStreamAvailableScans** (**DataStreamHandle** dataStreamHandle, **Int32** *availableScans)

  Get the number of available scans.

- UeiDaqAPI int **UeiDaqGetDataStreamTotalScans** (**DataStreamHandle** dataStreamHandle, **Int32** *totalScans)

  Get the total number of scans.

- UeiDaqAPI int **UeiDaqSetDataStreamOffset** (**DataStreamHandle** dataStreamHandle, **Int32** offset)

  Change the pointer position.

- UeiDaqAPI int **UeiDaqGetDataStreamRelativeTo** (**DataStreamHandle** dataStreamHandle, **tUeiDataStreamRelativeTo** *relativeTo)

  Get the reference of the offset.

- UeiDaqAPI int **UeiDaqSetDataStreamRelativeTo** (**DataStreamHandle** dataStreamHandle, **tUeiDataStreamRelativeTo** relativeTo)

  Set the reference of the offset.

- UeiDaqAPI int **UeiDaqGetDeviceName** (**DeviceHandle** deviceHandle, char *deviceName, int *deviceNameLength)

  Get the device name.

- UeiDaqAPI int **UeiDaqGetDeviceSerialNumber** (**DeviceHandle** deviceHandle, char *serialNumber, int *serialNumberLength)

  Get the serial number.

- UeiDaqAPI int **UeiDaqGetDeviceIndex** (**DeviceHandle** deviceHandle, **Int32** *index)

  Get the index.

- UeiDaqAPI int **UeiDaqGetDeviceNumberOfAIDifferentialChannels** (**DeviceHandle** deviceHandle, **Int32** *numChannels)

Get the number of Analog Input differential channels.

- UeiDaqAPI int **UeiDaqGetDeviceNumberOfAISingleEndedChannels** (**DeviceHandle** deviceHandle, **Int32** *numChannels)

  Get the number of Analog Input single-ended channels.

- UeiDaqAPI int **UeiDaqGetDeviceNumberOfAOChannels** (**DeviceHandle** deviceHandle, **Int32** *numChannels)

  Get the number of Analog Outout channels.

- UeiDaqAPI int **UeiDaqGetDeviceNumberOfDIChannels** (**DeviceHandle** deviceHandle, **Int32** *numChannels)

  Get the number of Digital Input channels.

- UeiDaqAPI int **UeiDaqGetDeviceNumberOfDOChannels** (**DeviceHandle** deviceHandle, **Int32** *numChannels)

  Get the number of Digital Output channels.

- UeiDaqAPI int **UeiDaqGetDeviceNumberOfCIChannels** (**DeviceHandle** deviceHandle, **Int32** *numChannels)

  Get the number of Counter Input channels.

- UeiDaqAPI int **UeiDaqGetDeviceNumberOfCOChannels** (**DeviceHandle** deviceHandle, **Int32** *numChannels)

  Get the number of Counter Output channels.

- UeiDaqAPI int **UeiDaqGetDeviceNumberOfSerialPorts** (**DeviceHandle** deviceHandle, **Int32** *numSerialPorts)

  Get the number of serial ports.

- UeiDaqAPI int **UeiDaqGetDeviceNumberOfSynchronousSerialPorts** (**DeviceHandle** deviceHandle, **Int32** *numSynchronousSerialPorts)

  Get the number of synchronous serial ports.

- UeiDaqAPI int **UeiDaqGetDeviceNumberOfCANPorts** (**DeviceHandle** deviceHandle, **Int32** *numCANPorts)

  Get the number of CAN ports.

- UeiDaqAPI int **UeiDaqGetDeviceNumberOfARINCInputPorts** (**DeviceHandle** deviceHandle, **Int32** *numARINCInputPorts)

  Get the number of ARINC-429 input ports.

- UeiDaqAPI int **UeiDaqGetDeviceNumberOfARINCOutputPorts** (**DeviceHandle** deviceHandle, **Int32** *numARINCOutputPorts)

  Get the number of ARINC-429 output ports.

- UeiDaqAPI int **UeiDaqGetDeviceLowRanges** (**DeviceHandle** deviceHandle, **tUeiSessionType** type, int subsystem, **Int32** *numValues, double *lowValues)

  Get the low ranges.

- UeiDaqAPI int **UeiDaqGetDeviceHighRanges** (**DeviceHandle** deviceHandle, **tUeiSessionType** type, int subsystem, **Int32** *numValues, double *highValues)

  Get the high ranges.

- UeiDaqAPI int **UeiDaqGetDeviceMaxAIRate** (**DeviceHandle** deviceHandle, **Int32** *maxRate)

  Get the maximum AI rate.

- UeiDaqAPI int **UeiDaqGetDeviceMaxAORate** (**DeviceHandle** deviceHandle, **Int32** *maxRate)

  Get the maximum AO rate.

- UeiDaqAPI int **UeiDaqGetDeviceMaxDIRate** (**DeviceHandle** deviceHandle, **Int32** *maxRate)

  Get the maximum DI rate.

- UeiDaqAPI int **UeiDaqGetDeviceMaxDORate** (**DeviceHandle** deviceHandle, **Int32** *maxRate)

  Get the maximum DO rate.

- UeiDaqAPI int **UeiDaqGetDeviceMaxCIRate** (**DeviceHandle** deviceHandle, **Int32** *maxRate)

  Get the maximum CI rate.

- UeiDaqAPI int **UeiDaqGetDeviceMaxCORate** (**DeviceHandle** deviceHandle, **Int32** *maxRate)

  Get the maximum CO rate.

- UeiDaqAPI int **UeiDaqGetDeviceAIResolution** (**DeviceHandle** deviceHandle, **Int32** *resolution)

  Get the AI resolution.

- UeiDaqAPI int **UeiDaqGetDeviceAOResolution** (**DeviceHandle** deviceHandle, **Int32** *resolution)

  Get the AO resolution.

- UeiDaqAPI int **UeiDaqGetDeviceDIResolution** (**DeviceHandle** deviceHandle, **Int32** *resolution)

  Get the DI resolution.

- UeiDaqAPI int **UeiDaqGetDeviceDOResolution** (**DeviceHandle** deviceHandle, **Int32** *resolution)

  Get the DO resolution.

- UeiDaqAPI int **UeiDaqGetDeviceCIResolution** (**DeviceHandle** deviceHandle, **Int32** *resolution)

  Get the CI resolution.

- UeiDaqAPI int **UeiDaqGetDeviceCOResolution** (**DeviceHandle** deviceHandle, **Int32** *resolution)

  Get the CO resolution.

- UeiDaqAPI int **UeiDaqGetDeviceAIDataSize** (**DeviceHandle** deviceHandle, **Int32** *dataSize)

  Get the AI data size.

- UeiDaqAPI int **UeiDaqGetDeviceAODataSize** (**DeviceHandle** deviceHandle, **Int32** *dataSize)

  Get the AI data size.

- UeiDaqAPI int **UeiDaqGetDeviceDIDataSize** (**DeviceHandle** deviceHandle, **Int32** *dataSize)

  Get the AI data size.

- UeiDaqAPI int **UeiDaqGetDeviceDODataSize** (**DeviceHandle** deviceHandle, **Int32** *dataSize)

  Get the AI data size.

- UeiDaqAPI int **UeiDaqGetDeviceCIDataSize** (**DeviceHandle** deviceHandle, **Int32** *dataSize)

  Get the AI data size.

- UeiDaqAPI int **UeiDaqGetDeviceCODataSize** (**DeviceHandle** deviceHandle, **Int32** *dataSize)

  Get the AI data size.

- UeiDaqAPI int **UeiDaqGetDeviceAISimultaneous** (**DeviceHandle** deviceHandle, **Int32** *simultaneous)

  Check if the device supports simultaneous sampling.

- UeiDaqAPI int **UeiDaqGetDeviceAOSimultaneous** (**DeviceHandle** deviceHandle, **Int32** *simultaneous)

  Check if the device supports simultaneous update.

- UeiDaqAPI int **UeiDaqGetDeviceInputFIFOSize** (**DeviceHandle** deviceHandle, **Int32** *inputFIFOSize)

  Get the size of the input FIFO.

- UeiDaqAPI int **UeiDaqGetDeviceOutputFIFOSize** (**DeviceHandle** deviceHandle, **Int32** *outputFIFOSize)

  Get the size of the output FIFO.

- UeiDaqAPI int **UeiDaqGetDeviceCanRegenerate** (**DeviceHandle** deviceHandle, **Int32** *canRegenerate)

  Check if the device supports output regeneration.

- UeiDaqAPI int **UeiDaqGetDeviceBidirectionalDigitalPorts** (**DeviceHandle** deviceHandle, **Int32** *isBidirectional)

  Check if the device digital ports are bidirectional.

- UeiDaqAPI int **UeiDaqGetDeviceSupportsDigitalFiltering** (**DeviceHandle** deviceHandle, **Int32** *digitalFiltering)

Check if the device supports digital filtering (debouncing).

- UeiDaqAPI int **UeiDaqGetDeviceHardwareInformation** (**DeviceHandle** deviceHandle, char *info, int *infoLength)

  Get the device hardware information.

- UeiDaqAPI int **UeiDaqGetDeviceStatus** (**DeviceHandle** deviceHandle, char *status, int *statusLength)

  Get the device status information.

- UeiDaqAPI int **UeiDaqReadDeviceEEPROM** (**DeviceHandle** deviceHandle, int bank, int subSystem, **uInt8** *buffer, int *size, **tUeiEEPROMArea** area)

  Read Device EEPROM.

- UeiDaqAPI int **UeiDaqWriteDeviceEEPROM** (**DeviceHandle** deviceHandle, int bank, int subSystem, int size, **uInt8** *buffer, **tUeiEEPROMArea** area, int saveEEPROM)

  Write Device EEPROM.

- UeiDaqAPI int **UeiDaqReadDeviceRegister32** (**DeviceHandle** deviceHandle, **uInt32** offset, **uInt32** *value)

  Read Device register.

- UeiDaqAPI int **UeiDaqWriteDeviceRegister32** (**DeviceHandle** deviceHandle, **uInt32** offset, **uInt32** value)

  Write Device register.

- UeiDaqAPI int **UeiDaqWriteReadDeviceRegisters32** (**DeviceHandle** deviceHandle, **uInt32** writeAddr, **uInt32** writeSize, **uInt32** *writeData, **uInt32** readAddr, **uInt32** readSize, **uInt32** *readData, int doReadFirst, int noIncrement)

  Performs a write and read of multiple values into the address space of the Device.

- UeiDaqAPI int **UeiDaqReadDeviceRAM** (**DeviceHandle** deviceHandle, **uInt32** address, int size, **uInt8** *buffer)

  Read from device RAM.

- UeiDaqAPI int **UeiDaqWriteDeviceRAM** (**DeviceHandle** deviceHandle, **uInt32** address, int size, **uInt8** *buffer)

  Write to device RAM.

- UeiDaqAPI int **UeiDaqGetDeviceTimeout** (**DeviceHandle** deviceHandle, **Int32** *pTimeout)

  Get the low-level access timeout.

- UeiDaqAPI int **UeiDaqSetDeviceTimeout** (**DeviceHandle** deviceHandle, **Int32** timeout)

  Set the low-level access timeout.

- UeiDaqAPI int **UeiDaqResetDevice** (**DeviceHandle** deviceHandle)

  Reset device.

- UeiDaqAPI int **UeiDaqSetDeviceWatchDogCommand** (**DeviceHandle** deviceHandle, **tUeiWatchDogCommand** cmd, int timeout)

Set watchdog command.

- UeiDaqAPI int **UeiDaqWriteDeviceCalibration** (**DeviceHandle** deviceHandle, int channel, **uInt8** dacMode, **uInt32** value, int saveEEPROM)

  Write calibration value in device's EEPROM.

- UeiDaqAPI int **UeiDaqGetTimingMode** (**TimingHandle** timingHandle, **tUeiTimingMode** *mode)

  Get the timing mode.

- UeiDaqAPI int **UeiDaqSetTimingMode** (**TimingHandle** timingHandle, **tUeiTimingMode** mode)

  Set the timing mode.

- UeiDaqAPI int **UeiDaqGetTimingConvertClockSource** (**TimingHandle** timingHandle, **tUeiTimingClockSource** *source)

  Get the Conversion clock source.

- UeiDaqAPI int **UeiDaqSetTimingConvertClockSource** (**TimingHandle** timingHandle, **tUeiTimingClockSource** source)

  Set the Conversion clock source.

- UeiDaqAPI int **UeiDaqGetTimingScanClockSource** (**TimingHandle** timingHandle, **tUeiTimingClockSource** *source)

  Get the Scan clock source.

- UeiDaqAPI int **UeiDaqSetTimingScanClockSource** (**TimingHandle** timingHandle, **tUeiTimingClockSource** source)

  Set the Scan clock source.

- UeiDaqAPI int **UeiDaqGetTimingConvertClockRate** (**TimingHandle** timingHandle, **f64** *rate)

  Get the conversion clock rate.

- UeiDaqAPI int **UeiDaqSetTimingConvertClockRate** (**TimingHandle** timingHandle, **f64** rate)

  Set the conversion clock rate.

- UeiDaqAPI int **UeiDaqGetTimingScanClockRate** (**TimingHandle** timingHandle, **f64** *rate)

  Get the scan clock rate.

- UeiDaqAPI int **UeiDaqSetTimingScanClockRate** (**TimingHandle** timingHandle, **f64** rate)

  Set the scan clock rate.

- UeiDaqAPI int **UeiDaqGetTimingConvertClockEdge** (**TimingHandle** timingHandle, **tUeiDigitalEdge** *edge)

  Get the conversion clock edge.

- UeiDaqAPI int **UeiDaqSetTimingConvertClockEdge** (**TimingHandle** timingHandle, **tUeiDigitalEdge** edge)

  Set the conversion clock edge.

- UeiDaqAPI int **UeiDaqGetTimingScanClockEdge** (**TimingHandle** timingHandle, **tUeiDigitalEdge** *edge)

  Get the scan clock edge.

- UeiDaqAPI int **UeiDaqSetTimingScanClockEdge** (**TimingHandle** timingHandle, **tUeiDigitalEdge** edge)

  Set the scan clock edge.

- UeiDaqAPI int **UeiDaqGetTimingDuration** (**TimingHandle** timingHandle, **tUeiTimingDuration** *duration)

  Get the duration.

- UeiDaqAPI int **UeiDaqSetTimingDuration** (**TimingHandle** timingHandle, **tUeiTimingDuration** duration)

  Set the duration.

- UeiDaqAPI int **UeiDaqGetTimingTimeout** (**TimingHandle** timingHandle, **Int32** *timeout)

  Get the timeout.

- UeiDaqAPI int **UeiDaqSetTimingTimeout** (**TimingHandle** timingHandle, **Int32** timeout)

  Set the timeout.

- UeiDaqAPI int **UeiDaqGetTimingScanClockTimebaseDivisor** (**TimingHandle** timingHandle, int *divisor)

  Get the scan clock divisor.

- UeiDaqAPI int **UeiDaqSetTimingScanClockTimebaseDivisor** (**TimingHandle** timingHandle, int divisor)

  Set the scan clock divisor.

- UeiDaqAPI int **UeiDaqGetTimingConvertClockTimebaseDivisor** (**TimingHandle** timingHandle, int *divisor)

  Get the convert clock divisor.

- UeiDaqAPI int **UeiDaqSetTimingConvertClockTimebaseDivisor** (**TimingHandle** timingHandle, int divisor)

  Set the convert clock divisor.

- UeiDaqAPI int **UeiDaqGetTimingScanClockTimebaseDividingCounter** (**TimingHandle** timingHandle, int *counter)

  Get the scan clock dividing counter.

- UeiDaqAPI int **UeiDaqSetTimingScanClockTimebaseDividingCounter** (**TimingHandle** timingHandle, int counter)

  Set the scan clock dividing counter.

- UeiDaqAPI int **UeiDaqTimingGetConvertClockTimebaseDividingCounter** (**TimingHandle** timingHandle, int *counter)

  Get the convert clock dividing counter.

- UeiDaqAPI int **UeiDaqSetTimingConvertClockTimebaseDividingCounter** (**TimingHandle** timingHandle, int counter)

  Set the convert clock dividing counter.

- UeiDaqAPI int **UeiDaqGetTimingScanClockSourceSignal** (**TimingHandle** timingHandle, char *signal, int *signalLength)

  Get the name of the signal to use as Scan clock.

- UeiDaqAPI int **UeiDaqSetTimingScanClockSourceSignal** (**TimingHandle** timingHandle, char *signal)

  Set the name of the signal to use as Scan clock.

- UeiDaqAPI int **UeiDaqGetTimingScanClockDestinationSignal** (**TimingHandle** timingHandle, char *signal, int *signalLength)

  Get the name of the signal to route the Scan clock to.

- UeiDaqAPI int **UeiDaqSetTimingScanClockDestinationSignal** (**TimingHandle** timingHandle, char *signal)

  Set the name of the signal to route the Scan clock to.

- UeiDaqAPI int **UeiDaqGetTimingConvertClockSourceSignal** (**TimingHandle** timingHandle, char *signal, int *signalLength)

  Get the name of the signal to use as conversion clock.

- UeiDaqAPI int **UeiDaqSetTimingConvertClockSourceSignal** (**TimingHandle** timingHandle, char *signal)

  Set the name of the signal to use as conversion clock.

- UeiDaqAPI int **UeiDaqGetTimingConvertClockDestinationSignal** (**TimingHandle** timingHandle, char *signal, int *signalLength)
- UeiDaqAPI int **UeiDaqSetTimingConvertClockDestinationSignal** (**TimingHandle** timingHandle, char *signal)

  Set the name of the signal to route the conversion clock to.

- UeiDaqAPI int **UeiDaqGetTriggerSource** (**TriggerHandle** triggerHandle, **tUeiTriggerSource** *source)

  Get the trigger source.

- UeiDaqAPI int **UeiDaqSetTriggerSource** (**TriggerHandle** triggerHandle, **tUeiTriggerSource** source)

  Set the trigger source.

- UeiDaqAPI int **UeiDaqGetTriggerDigitalEdge** (**TriggerHandle** triggerHandle, **tUeiDigitalEdge** *edge)

  Get the trigger edge.

- UeiDaqAPI int **UeiDaqSetTriggerDigitalEdge** (**TriggerHandle** triggerHandle, **tUeiDigitalEdge** edge)

  Set the trigger edge.

- UeiDaqAPI int **UeiDaqGetTriggerSourceSignal** (**TriggerHandle** triggerHandle, char *signal, int *signalLength)

  Get the trigger source signal.

- UeiDaqAPI int **UeiDaqSetTriggerSourceSignal** (**TriggerHandle** triggerHandle, char *signal)

  Set the trigger source signal.

- UeiDaqAPI int **UeiDaqGetTriggerDestinationSignal** (**TriggerHandle** triggerHandle, char *signal, int *signalLength)

  Get the trigger destination signal.

- UeiDaqAPI int **UeiDaqSetTriggerDestinationSignal** (**TriggerHandle** triggerHandle, char *signal)

  Set the trigger destination signal.

- UeiDaqAPI int **UeiDaqGetTriggerAction** (**TriggerHandle** triggerHandle, **tUeiTriggerAction** *pAction)

  Get the trigger action.

- UeiDaqAPI int **UeiDaqSetTriggerAction** (**TriggerHandle** triggerHandle, **tUeiTriggerAction** action)

  Set the trigger action.

- UeiDaqAPI int **UeiDaqGetTriggerCondition** (**TriggerHandle** triggerHandle, **tUeiTriggerCondition** *pCondition)

  Get the analog trigger condition.

- UeiDaqAPI int **UeiDaqSetTriggerCondition** (**TriggerHandle** triggerHandle, **tUeiTriggerCondition** condition)

  Set the analog trigger condition.

- UeiDaqAPI int **UeiDaqGetTriggerChannel** (**TriggerHandle** triggerHandle, **Int32** *pChannel)

  Get the trigger channel.

- UeiDaqAPI int **UeiDaqSetSetTriggerChannel** (**TriggerHandle** triggerHandle, **Int32** channel)

  Set the trigger channel.

- UeiDaqAPI int **UeiDaqGetTriggerLevel** (**TriggerHandle** triggerHandle, **f64** *pLevel)

  Get the trigger level.

- UeiDaqAPI int **UeiDaqSetTriggerLevel** (**TriggerHandle** triggerHandle, **f64** level)

  Set the trigger level.

- UeiDaqAPI int **UeiDaqGetTriggerHysteresis** (**TriggerHandle** triggerHandle, **f64** *pHysteresis)

  Get the trigger hysteresis.

- UeiDaqAPI int **UeiDaqSetTriggerHysteresis** (**TriggerHandle** triggerHandle, **f64** hysteresis)

  Set the trigger hysteresis.

- UeiDaqAPI int **UeiDaqGetNumberOfPreTriggerScans** (**TriggerHandle** triggerHandle, **Int32** *pNumPreTriggerScans)

  Get the number of pre-trigger scans.

- UeiDaqAPI int **UeiDaqSetNumberOfPreTriggerScans** (**TriggerHandle** triggerHandle, **Int32** numPreTriggerScans)

  Set the number of pre-trigger scans.

- UeiDaqAPI int **UeiDaqFireTrigger** (**TriggerHandle** triggerHandle)

  Manually send a trigger.

- UeiDaqAPI const char * **UeiDaqTranslateError** (int error)

  Translate error code.

- UeiDaqAPI int **UeiDaqEnumDriver** (int index, char *driverName)

  Enumerate drivers.

- UeiDaqAPI int **UeiDaqEnumDevice** (char *deviceResourceName, **DeviceHandle** *deviceHandle)

  Enumerate devices.

- UeiDaqAPI int **UeiDaqParseResource** (char *resource, char *deviceClass, char *remoteAddress, int *deviceID, **tUeiSessionType** *type, int *channelList, int *channelListSize)

  Parse resource string.

- UeiDaqAPI int **UeiDaqSetSessionCustomProperty** (**SessionHandle** sessionHandle, char *property, int valueSize, void *value)

  Set a custom property.

- UeiDaqAPI int **UeiDaqGetSessionCustomProperty** (**SessionHandle** sessionHandle, char *property, int valueSize, void *value)

  Get a custom property.

- UeiDaqAPI int **UeiDaqGetSessionType** (**SessionHandle** sessionHandle, **tUeiSessionType** *sessionType)

  Get the session type.

- UeiDaqAPI int **UeiDaqLoadSessionFromXml** (**SessionHandle** sessionHandle, char *xmlSettings)

  Load Session settings from an XML stream.

- UeiDaqAPI int **UeiDaqLoadSessionFromFile** (**SessionHandle** sessionHandle, char *sessionFile)

Load Session settings from a file.

- UeiDaqAPI int **UeiDaqGetFrameworkVersion** (char *version, int versionSize)
  Get the framework version string.
- UeiDaqAPI int **UeiDaqGetPluginLowLevelDriverVersion** (char *driverName, char *version, int versionSize)
  Get a driver plugin version string.
- UeiDaqAPI int **UeiDaqGetFrameworkInstallationDirectory** (char *installDir, int installDirSize)
  Get the framework installation directory.
- UeiDaqAPI int **UeiDaqGetPluginsInstallationDirectory** (char *installDir, int installDirSize)
  Get the framework plugins directory.
- UeiDaqAPI int **UeiDaqGetSessionGroupDirectory** (char *sessionGroup, char *sessionGroupDir, int sessionGroupDirSize)
  Get a session group directory.
- UeiDaqAPI **tUeiOSType UeiDaqGetOperatingSystemType** ()
  Get the operating system type.
- UeiDaqAPI void **UeiDaqReloadDrivers** ()
  Reload driver plugins.

3.2.1 Detailed Description

The UeiDaq Framework ANSI-C programming interface allows you to program the framework using the ANSI-C language. It is very close to the UeiDaq C++ class library model. Each framework C++ class is represented in ANSI-C by a handle datatype and there is an ANSI-C funtion corresponding to each C++ class method.

3.2.2 Typedef Documentation

3.2.2.1 typedef void* ChannelHandle

Call UeiDaqGetChannelHandle to get the handle of one of the channel objects associated with a session.

3.2.2.2 typedef void* DataStreamHandle

Call UeiDaqGetDataStreamHandle to get the handle of the DataStream object associated with a session.

3.2.2.3 typedef void* DeviceHandle

Call UeiDaqGetDeviceHandle to get the handle of the device associated with a session.

3.2.2.4 typedef void* SessionHandle

Represents a session object created with UeiDaqCreateSession and destroyed with UeiDaqCloseSession.

3.2.2.5 typedef void* TimingHandle

Call UeiDaqGetTimingHandle to get the handle of the timing object associated with a session.

3.2.2.6 typedef void* TriggerHandle

Call UeiDaqGet**TriggerHandle to get the handle of one of the trigger objects associated with a session.

3.2.2.7 typedef void(*) tUeiEventCallback(tUeiEvent event, void *param)

The prototype of the callback function called when reading/writing asynchronously. You can initiate an asynchronous operation using one of the UeiDaqRead**Async or UeiDaqWrite***Async function calls.

3.2.3 Function Documentation

3.2.3.1 UeiDaqAPI int SetDIIndustrialVoltageSupply (ChannelHandle *channelHandle*, int *line*, double *voltage*)

Program the voltage supplied to selected line. Test mode: Guardian feature that connects an internal voltage source to each input line in order to test that it is functional Pull-up mode: connect a pull-up resistor between the internal voltage source and the input line to monitor switch or contacts without external circuitry.

Parameters:

channelHandle The handle to an existing industrial DI channel

line The input line to configure

voltage the test voltage

Returns:

The status code.

3.2.3.2 UeiDaqAPI int UeiDaqAddARINCInputFilterEntry (ChannelHandle *channelHandle*, tUeiARINCFilterEntry *entry*)

Add a filter entry to the port's frame filter table. Each incoming ARINC frame that doesn't match any of the filter entries is rejected.

Parameters:

channelHandle The handle to an existing ARINC-429 input channel

entry A struture that represents the new filter entry

Returns:

The status code.

See also:

UeiDaqClearARINCInputFilterEntries (p. 707), **UeiDaqGetARINCInputFilterEntry** (p. 769)

3.2.3.3 UeiDaqAPI int UeiDaqAddARINCOutputMinorFrameEntry (ChannelHandle *channelHandle*, tUeiARINCMinorFrameEntry *entry*)

Add a minor frame entry to the port's scheduler table.

Parameters:

channelHandle The handle to an existing ARINC-429 output channel

entry A struture that represents the new minor frame entry

Returns:

The status code.

See also:

UeiDaqClearARINCOutputMinorFrameEntries (p. 707), **UeiDaqGetARINCOutputMinorFrameEntry** (p. 771)

3.2.3.4 UeiDaqAPI int UeiDaqAddARINCOutputSchedulerEntry (ChannelHandle *channelHandle*, tUeiARINCSchedulerEntry *entry*)

Add a scheduler entry to the port's scheduler table.

Parameters:

channelHandle The handle to an existing ARINC-429 output channel

entry A struture that represents the new scheduler entry

Returns:

The status code.

See also:

UeiDaqClearARINCOutputSchedulerEntries (p. 708), **UeiDaqGetARINCOutputSchedulerEntry** (p. 771)

3.2.3.5 UeiDaqAPI int UeiDaqAddCANPortFilterEntry (ChannelHandle *channelHandle*, tUeiCANFilterEntry *entry*)

Add a filter entry to the port's filter table. Each incoming CAN frame that doesn't match any of the filter ranges is rejected.

Parameters:

channelHandle The handle to an existing CAN port

entry A struture that represents the new filter entry

Returns:

The status code.

See also:

UeiDaqClearCANPortFilterEntries (p. 708), **UeiDaqGetCANPortFilterEntry** (p. 775)

3.2.3.6 UeiDaqAPI int UeiDaqAddMIL1553FilterEntry (ChannelHandle *channelHandle*, tUeiMIL1553FilterEntry *entry*)

Add a filter entry to the port's frame filter table. Each incoming MIL-1553 frame that doesn't match any of the filter entries is rejected.

Parameters:

channelHandle The handle to an existing MIL-1553 channel

entry A struture that represents the new filter entry

Returns:

The status code.

See also:

UeiDaqClearMIL1553FilterEntries (p. 708), **UeiDaqGetMIL1553FilterEntry** (p. 819)

3.2.3.7 UeiDaqAPI int UeiDaqAddMIL1553SchedulerEntry (ChannelHandle *channelHandle*, tUeiMIL1553SchedulerEntry *entry*)

Add a scheduler entry to the port's scheduler table.

Parameters:

channelHandle The handle to an existing MIL-1553 channel

entry A struture that represents the new scheduler entry

Returns:

The status code.

See also:

UeiDaqClearMIL1553SchedulerEntries (p. 709), **UeiDaqGetMIL1553SchedulerEntry** (p. 820)

3.2.3.8 UeiDaqAPI int UeiDaqCleanUpSession (SessionHandle *sessionHandle*)

Stop and Clean-up the session. All session parameters are lost and you need to reconfigure channels, timing or triggers if you want to reuse the same session object.

Parameters:

sessionHandle The handle to an existing session

Returns:

The status code

See also:

UeiDaqStopSession (p. 955) **UeiDaqStartSession** (p. 955) **UeiDaqIsSessionRunning** (p. 865) **UeiDaqWaitUntilSessionIsDone** (p. 957)

3.2.3.9 UeiDaqAPI int UeiDaqClearARINCInputFilterEntries (ChannelHandle *channelHandle*)

Empties the port's filter table.

Parameters:

channelHandle The handle to an existing ARINC-429 input channel

Returns:

The status code.

See also:

UeiDaqAddARINCInputFilterEntry (p. 704), **UeiDaqGetARINCInputFilterEntry** (p. 769)

3.2.3.10 UeiDaqAPI int UeiDaqClearARINCOutputMinorFrameEntries (ChannelHandle *channelHandle*)

Empties the port's major frame.

Parameters:

channelHandle The handle to an existing ARINC-429 output channel

Returns:

The status code.

See also:

UeiDaqAddARINCOutputMinorFrameEntry (p. 705), **UeiDaqGetARINCOutputMinorFrameEntry** (p. 771)

3.2.3.11 UeiDaqAPI int UeiDaqClearARINCOutputSchedulerEntries (ChannelHandle *channelHandle*)

Empties the port's scheduler table.

Parameters:

channelHandle The handle to an existing ARINC-429 output channel

Returns:

The status code.

See also:

UeiDaqAddARINCOutputSchedulerEntry (p. 705), **UeiDaqGetARINCOutputSchedulerEntry** (p. 771)

3.2.3.12 UeiDaqAPI int UeiDaqClearCANPortFilterEntries (ChannelHandle *channelHandle*)

Empties the port's filter table.

Parameters:

channelHandle The handle to an existing CAN port

Returns:

The status code.

See also:

UeiDaqAddCANPortFilterEntry (p. 706), **UeiDaqGetCANPortFilterEntry** (p. 775)

3.2.3.13 UeiDaqAPI int UeiDaqClearMIL1553FilterEntries (ChannelHandle *channelHandle*)

Empties the port's filter table.

Parameters:

channelHandle The handle to an existing MIL-1553 channel

Returns:

The status code.

See also:

UeiDaqAddMIL1553FilterEntry (p. 706), **UeiDaqGetMIL1553FilterEntry** (p. 819)

3.2.3.14 UeiDaqAPI int UeiDaqClearMIL1553SchedulerEntries (ChannelHandle *channelHandle*)

Empties the port's scheduler table.

Parameters:

channelHandle The handle to an existing MIL-1553 channel

Returns:

The status code.

See also:

UeiDaqAddMIL1553SchedulerEntry (p. 706), **UeiDaqGetMIL1553SchedulerEntry** (p. 820)

3.2.3.15 UeiDaqAPI int UeiDaqCloseDevice (DeviceHandle *deviceHandle*)

Releases resources for the specified device. Call this function to release handles obtained with UeiDaqEnumDevice

Parameters:

deviceHandle The handle to close

Returns:

The status code

3.2.3.16 UeiDaqAPI int UeiDaqCloseSession (SessionHandle *sessionHandle*)

Releases resources for the specified session.

Parameters:

sessionHandle The handle to the session

Returns:

The status code

3.2.3.17 UeiDaqAPI int UeiDaqConfigureAnalogSoftwareTrigger (SessionHandle *sessionHandle*, tUeiTriggerAction *action*, tUeiTriggerCondition *condition*, Int32 *triggerChannel*, f64 *level*, f64 *hysteresis*, Int32 *preTriggerScans*)

The analog software trigger looks at every sample acquired on the specified channel until the trigger condition is met. Once the trigger condition is met the application can read the acquired scans in a buffer. The trigger level and hysteresis are specified in the same unit as the measurements.

Parameters:

sessionHandle The handle to an existing session

action The action to execute when the trigger condition is met

condition The condition for the trigger to occur.

triggerChannel The analog input channel to monitor for the trigger condition.

level The scaled value that defines the trigger threshold.

hysteresis The scaled size of the hysteresis window.

preTriggerScans The number of scans to save before the trigger occurrence.

Returns:

The status code

3.2.3.18 UeiDaqAPI int UeiDaqConfigureSignalTrigger (SessionHandle *sessionHandle*, tUeiTriggerAction *action*, char * *signal*)

Configure the trigger signal that will start or stop the session The trigger signal is device dependent, it can be a physical line on a backplane (such as a PXI trigger) or a software signal used to synchronize multiple devices (for example starting all layers in a PowerDNA cube simultaneously).

Parameters:

sessionHandle The handle to an existing session

action The action to execute when the signal occurs.

signal The signal that will transmit the trigger

Returns:

The status code

3.2.3.19 UeiDaqAPI int UeiDaqConfigureStartDigitalTrigger (SessionHandle *sessionHandle*, tUeiTriggerSource *source*, tUeiDigitalEdge *edge*)

Configure the digital trigger condition that will start the session

Parameters:

sessionHandle The handle to an existing session

source The trigger source descriptor

edge The edge of the trigger signal that starts the session.

Returns:

The status code

3.2.3.20 UeiDaqAPI int UeiDaqConfigureStopDigitalTrigger (SessionHandle *sessionHandle*, tUeiTriggerSource *source*, tUeiDigitalEdge *edge*)

Configure the digital trigger condition that will stop the session

Parameters:

sessionHandle The handle to an existing session

source The trigger source descriptor

edge The edge of the trigger signal that stops the session.

Returns:

The status code

3.2.3.21 UeiDaqAPI int UeiDaqConfigureTimingForAsynchronousIO (SessionHandle *sessionHandle*, int *watermark*, int *periodUs*, int *noActivityTimeoutUs*, int *maxDataSize*)

Configure the timing object for asynchronous FIFO based I/O Data is read/written when the input/output FIFO reaches the specified watermark level, when the periodic timer expires or when the no activity timeout expires (whichever happens first) Set watermark to -1 to disable watermark events Set period to -1 to disable periodic timer events Set no activity timeout to -1 to disable no activity events

Parameters:

sessionHandle The handle to an existing session

watermark The watermark level used to configure FIFO asynchronous event. (-1) to disable

periodUs The period at which an event is fired when the FIFO level stays below watermark for too long (-1) to disable)

noActivityTimeoutUs The maximum time to wait for any activity (-1) disable

maxDataSize The maximum number of values returned along with an event

Returns:

The status code

3.2.3.22 UeiDaqAPI int UeiDaqConfigureTimingForBufferedIO (SessionHandle *sessionHandle*, int *samplePerChannel*, tUeiTimingClockSource *clkSource*, double *rate*, tUeiDigitalEdge *edge*, tUeiTimingDuration *duration*)

Configure the timing object for buffered mode using hardware timing

Parameters:

sessionHandle The handle to an existing session

samplePerChannel The size of the buffer used to transfer sample from/to the device.

clkSource The source of the clock.

rate The frequency of the hardware clock pacing the acquisition/generation of each scan in Hz.

edge The edge of the clock that triggers the scan acquisition/generation: rising or falling.

duration The duration of the acquisition/generation, continuous or single-shot.

Returns:

The status code

See also:

UeiDaqConfigureTimingForSimpleIO (p. 713), **UeiDaqConfigureTimingForEdgeDetection** (p. 712), **UeiDaqConfigureTimingForTimeSequencing** (p. 713)

3.2.3.23 UeiDaqAPI int UeiDaqConfigureTimingForDataMappingIO (SessionHandle *sessionHandle*, tUeiTimingClockSource *clkSource*, double *rate*)

Configure the timing object for direct data mapping mode using hardware timing

Parameters:

sessionHandle The handle to an existing session

clkSource The source of the clock.

rate The refresh rate of the data map.

See also:

UeiDaqConfigureTimingForSimpleIO (p. 713), **UeiDaqConfigureTimingForEdgeDetection** (p. 712), **UeiDaqConfigureTimingForTimeSequencing** (p. 713)

3.2.3.24 UeiDaqAPI int UeiDaqConfigureTimingForEdgeDetection (SessionHandle *sessionHandle*, tUeiDigitalEdge *edge*)

Configure the timing object for digital line state change detection The line on which the state change is sensed is specified by calling **UeiDaqSetDIChannelEdgeMask()** (p. 916).

Parameters:

sessionHandle The handle to an existing session

edge The edge of the signal that triggers the state change detection event.

Returns:

The status code

See also:

UeiDaqSetDIChannelEdgeMask (p. 916), **UeiDaqConfigureTimingForSimpleIO** (p. 713), **UeiDaqConfigureTimingForBufferedIO** (p. 711), **UeiDaqConfigureTimingForTimeSequencing** (p. 713)

3.2.3.25 UeiDaqAPI int UeiDaqConfigureTimingForMessagingIO (SessionHandle *sessionHandle*, int *bufferSize*, double *refreshRate*)

Configure the session to do messaging IOs. This timing mode is only available for Serial and CAN bus sessions.

The serial and CAN bus devices can be programmed to wait for a certain number of messages to be received before notifying the session. It is also possible to program the maximum amount of time to wait for the specified number of messages before notifying the session.

For serial sessions a message is simply a byte, for CAN sessions a message is a CAN frame represented with the data structure tUeiCANFrame.

Parameters:

sessionHandle The handle to an existing session

bufferSize The miminum number of messages to buffer before notifying the session

refreshRate The rate at which the device notifies the session that messages have been received. Set the rate to 0 to be notified immediately when a message is received.

See also:

UeiDaqConfigureTimingForBufferedIO (p. 711), **UeiDaqConfigureTimingForEdgeDetection** (p. 712), **UeiDaqConfigureTimingForTimeSequencing** (p. 713)

3.2.3.26 UeiDaqAPI int UeiDaqConfigureTimingForSimpleIO (SessionHandle *sessionHandle*)

Configure the timing object for simple IO. In this mode scans are acquired/generated one at a time. The pace of the acquisition/generation is entirely determined by the speed at which your software is calling the read/write functions.

Parameters:

sessionHandle The handle to an existing session

Returns:

The status code

See also:

UeiDaqConfigureTimingForBufferedIO (p. 711), **UeiDaqConfigureTimingForEdgeDetection** (p. 712), **UeiDaqConfigureTimingForTimeSequencing** (p. 713)

3.2.3.27 UeiDaqAPI int UeiDaqConfigureTimingForTimeSequencing (SessionHandle *sessionHandle*, int *samplePerChannel*, tUeiTimingDuration *duration*)

In Time sequencing mode, the timing information is built-in the data buffer written to the board

Parameters:

sessionHandle The handle to an existing session

samplePerChannel The number of samples to acquire if duration is one-shot.

duration The duration of the acquisition/generation, continuous or single-shot.

Returns:

The status code

See also:

UeiDaqConfigureTimingForSimpleIO (p. 713), **UeiDaqConfigureTimingForEdgeDetection** (p. 712), **UeiDaqConfigureTimingForBufferedIO** (p. 711)

3.2.3.28 UeiDaqAPI int UeiDaqConfigureTimingForVMapIO (SessionHandle *sessionHandle*, double *rate*)

Configure the timing object forVMAP mode VMap mode provides access to a device's FIFO without any buffering

Parameters:

sessionHandle The handle to an existing session

rate The rate at which data is clocked in or out of the FIFO (relevant only for clocked I/Os such as AI, AO, DI and DO).

See also:

ConfigureTimingForSimpleIO, ConfigureTimingForEdgeDetection, ConfigureTimingForTimeSequencing

3.2.3.29 UeiDaqAPI int UeiDaqCreateAccelChannel (SessionHandle *sessionHandle*, char * *resource*, f64 *min*, f64 *max*, double *sensorSensitivity*, double *excitationCurrent*, tUeiCoupling *coupling*, int *enableLowPassFilter*)

Create and add one or more accelerometer channel to the channel list associated with the session. This will only work with devices that can provide excitation current for ICP and IEPE sensors.

Example:

```
SessionHandle mySession;
UeiDaqCreateSession(&mySession);

// Configure channel 0 on device 1 to acquire acceleration measured
// by a sensor with a sensitivity of 24 mV/g, powered by a current
// of 5mA. the gain of the device is adjusted to measure accelerations
// between -10.0g and + 10.0g
UeiDaqCreateAccelChannel(mySession, "pwrdaq://Dev1/Ai0", -10.0, 10.0, 24,
                         5.0, UeiCouplingAC);
```

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

min The minimum value you expect to measure. The unit depends on the unit of the accelerometer sensitivity. (mV/g will provide measurements in g).

max The maximum value you expect to measure. The unit depends on the unit of the accelerometer sensitivity. (mV/g will provide measurements in g).

sensorSensitivity The sensitivity of the accelerometer. The unit of this parameter specifies the unit of the measurements.

excitationCurrent The excitation current to output to the sensor

coupling The coupling type. AC coupling turns on a 0.1Hz high pass filter.

enableLowPassFilter Turn on or off the low-pass anti-aliasing filter

Returns:

The status code.

See also:

UeiDaqCreateAIChannel (p. 715)

3.2.3.30 UeiDaqAPI int UeiDaqCreateAIChannel (SessionHandle *sessionHandle*, char * *resource*, f64 *min*, f64 *max*, tUeiAIChannelInputMode *mode*)

Create and add one or more channel to the AI channel list associated with the session.

Example:

```
SessionHandle mySession;
UeiDaqCreateSession(&mySession);
UeiDaqCreateAIChannel(mySession, "pwrdaq://Dev1/Ai0", -10.0, 10.0, UeiAIChannelInputModeDifferential );
```

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

min The minimum value you expect to measure.

max The maximum value you expect to measure

mode The input mode of the Analog input(s)

Returns:

The status code

3.2.3.31 UeiDaqAPI int UeiDaqCreateAICurrentChannel (SessionHandle *sessionHandle*, char * *resource*, f64 *min*, f64 *max*, tUeiFeatureEnable *enableCB*, tUeiAIChannelInputMode *mode*)

Create and add one or more channel to the AI channel list associated with the session.

Example:

```
SessionHandle mySession;
UeiDaqCreateSession(&mySession);
UeiDaqCreateAIChannel(mySession, "pwrdaq://Dev1/Ai0", -10.0, 10.0, UeiAIChannelInputModeDifferential );
```

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

min The minimum value you expect to measure.

max The maximum value you expect to measure

enableCB The circuit breaker configuration

mode The input mode of the Analog input(s)

Returns:

The status code

3.2.3.32 UeiDaqAPI int UeiDaqCreateAIVExChannel (SessionHandle *sessionHandle*, char * *resource*, f64 *min*, f64 *max*, tUeiSensorBridgeType *sensorBridgeType*, double *excitationVoltage*, int *useExcitationForScaling*, tUeiAIChannelInputMode *mode*)

Create and add one or more analog input channel to the channel list associated with the session. This will only work with devices that can provide excitation voltage.

Example:

```
SessionHandle mySession;
UeiDaqCreateSession(&mySession);
UeiDaqCreateAIVExChannel(mySession, "pdna://Dev1/Ai0", -100.0, 100.0, UeiSensorFullBridge,
                         10.0, true);
```

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

min The minimum value you expect to measure.

max The maximum value you expect to measure.

sensorBridgeType The type of the wheatstone bridge built-in the sensor

excitationVoltage The excitation voltage to output to the sensor

useExcitationForScaling Specifies whether the acquired data is divided by the excitation voltage

mode The input mode used to measure voltage from the sensor

Returns:

The status code.

See also:

UeiDaqCreateAIChannel (p. 715)

3.2.3.33 UeiDaqAPI int UeiDaqCreateAOChannel (SessionHandle *sessionHandle*, char * *resource*, f64 *min*, f64 *max*)

Create and add one or more channel to the AO channel list associated with the session.

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

min The minimum value you expect to generate.

max The maximum value you expect to generate

resource The device and channel(s) to add to the list.

Returns:

The status code

3.2.3.34 UeiDaqAPI int UeiDaqCreateAOCurrentChannel (SessionHandle *sessionHandle*, char * *resource*, f64 *min*, f64 *max*)

Create and add one or more current channel to the AO current channel list associated with the session. Current channels are only available on AO devices capable of outputing current

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

min The minimum value you expect to generate.

max The maximum value you expect to generate

Returns:

The status code

3.2.3.35 UeiDaqAPI int UeiDaqCreateAOProtectedChannel (SessionHandle *sessionHandle*, char * *resource*, tUeiAODACMode *mode*, double *measurementRate*, int *autoRetry*, double *retryRate*)

Create and add one or more channel to the channel list associated with the session. Protected AO channels are available on certain devices such as the DNA-AO-318 and DNA-AO-316. The amount of current flowing through each output is monitored at the given rate and must stay within the specified range, otherwise the device will automatically open the circuit acting as a breaker. You can specify whether the device should attempt to reestablish the circuit and how often it should try to do so.

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

mode Selects which DAC is conencted to the output

measurementRate The monitoring rate. Determines how fast the breaker react after an over or under-limit condition.

autoRetry Specifies whether the device will attempt to reestablish the circuit after an over or under-limit condition

retryRate The number of retries per second.

Returns:

A pointer to the first channel created.
The status code

3.2.3.36 UeiDaqAPI int UeiDaqCreateAOProtectedCurrentChannel (SessionHandle *sessionHandle*, char * *resource*, tUeiAODACMode *mode*, double *measurementRate*, int *autoRetry*, double *retryRate*)

Create and add one or more channel to the channel list associated with the session. Protected AO current channels are available on certain devices such as the DNA-AO-318-020 The amount of current flowing through each output is monitored at the given rate and must stay within the specified range, otherwise the device will automatically open the circuit acting as a breaker. You can specify whether the device should attempt to reestablish the circuit and how often it should try to do so.

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

mode Selects which DAC is connected to the output

measurementRate The monitoring rate. Determines how fast the breaker react after an over or under-limit condition.

autoRetry Specifies whether the device will attempt to reestablish the circuit after an over or under-limit condition

retryRate The number of retries per second.

Returns:

The status code

3.2.3.37 UeiDaqAPI int UeiDaqCreateAOWaveformChannel (SessionHandle *sessionHandle*, char * *resource*, tUeiAOWaveformClockSource *dacClockSource*, tUeiAOWaveformOffsetClockSource *offsetDACClockSource*, tUeiAOWaveformClockSync *clockSync*)

Create and add one or more waveform channel to the channel list associated with the session.

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

dacClockSource the main DAC clock source

offsetDACClockSource the offset DAC clock source

clockSync channel0 clock routing

Returns:

The status code

3.2.3.38 UeiDaqAPI int UeiDaqCreateARINCInputPort (SessionHandle *sessionHandle*, char * *resource*, tUeiARINCPortSpeed *bitsPerSecond*, tUeiARINCPortParity *parity*, int *enableSDIFilter*, uInt32 *SDIMask*)

Create and add one or more ARINC input ports to the session.

Parameters:

sessionHandle The handle to an existing session

resource The device and port(s) to add to the session.

bitsPerSecond The number of bits transmitted per second over the ARINC port

parity The parity used to detect transmission errors.

enableSDIFilter Set to true to enable frame filtering based on their SDI bits.

SDIMask The mask used to match incoming frame SDI bits (only bits 0 and 1 are significant).

Returns:

the status code.

See also:

UeiDaqCreateARINCOutputPort (p. 719)

3.2.3.39 UeiDaqAPI int UeiDaqCreateARINCOutputPort (SessionHandle *sessionHandle*, char * *resource*, tUeiARINCPortSpeed *bitsPerSecond*, tUeiARINCPortParity *parity*)

Create and add one or more ARINC output ports to the session.

Parameters:

sessionHandle The handle to an existing session

resource The device and port(s) to add to the session.

bitsPerSecond The number of bits transmitted per second over the ARINC port

parity The parity used to detect transmission errors.

Returns:

the status code.

See also:

UeiDaqCreateARINCInputPort (p. 719)

3.2.3.40 UeiDaqAPI int UeiDaqCreateCANPort (SessionHandle *sessionHandle*, char * *resource*, tUeiCANPortSpeed *bitsPerSecond*, tUeiCANFrameFormat *frameFormat*, tUeiCANPortMode *mode*, uInt32 *acceptanceMask*, uInt32 *acceptanceCode*)

Create and add one or more CAN ports to the session.

Parameters:

sessionHandle The handle to an existing session

resource The device and port(s) to add to the session.

bitsPerSecond The number of bits transmitted per second over the CAN port

frameFormat Specifies the format of frames sent, basic (11 bits ID) or extended (29 bits ID)

mode The operation mode, normal or passive to only listen

acceptanceMask The acceptance mask is used to filter incoming frames. The mask selects which bits within arbitration ID will be used for filtering.

acceptanceCode The acceptance code is used to filter incoming frames. The arbitration ID bits selected by the mask are compared to the code and the frame is rejected if there is any difference. If (mask XOR id) AND code == 0 the frame is accepted.

Returns:

the status code.

See also:

UeiDaqCreateSerialPort (p. 730)

3.2.3.41 UeiDaqAPI int UeiDaqCreateCIChannel (SessionHandle *sessionHandle*, char * *resource*, tUeiCounterSource *source*, tUeiCounterMode *mode*, tUeiCounterGate *gate*, Int32 *divider*, Int32 *inverted*)

Create and add one or more channel to the CI channel list associated with the session.

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

source The source of the signal counted or used as a timebase

mode The mode that specify whether the session counts events, measures a pulse width or measures a period on the signal configured as the source

gate The signal that specify whether the counter/timer is on or off

divider The divider used to scale the timebase speed

inverted Specifies whether the signal at the source is inverted before performing the counting operation

Returns:

The status code

3.2.3.42 UeiDaqAPI int UeiDaqCreateCOChannel (SessionHandle *sessionHandle*, char * *resource*, tUeiCounterSource *source*, tUeiCounterMode *mode*, tUeiCounterGate *gate*, uInt32 *tick1*, uInt32 *tick2*, Int32 *divider*, Int32 *inverted*)

Create and add one or more channel to the CO channel list associated with the session.

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

source The timebase used to determine the shape of the signal generated on the counter output

mode The mode that specify whether the session generate a pulse or a pulse train on the counter output

gate The signal that specify whether the counter/timer is on or off

tick1 Specifies how long the output stays low, number of ticks of the signal connected at the source.

tick2 Specifies how long the output stays high, number of ticks of the signal connected at the source.

divider The divider used to scale the timebase speed

inverted Specifies whether the signal at the source is inverted before performing the operation

Returns:

The status code

3.2.3.43 UeiDaqAPI int UeiDaqCreateCSDBPort (SessionHandle *sessionHandle*, char * *resource*, int *bps*, int *parity*, int *blockSize*, int *numberOfMessages*, int *interByteDelayUs*, int *interBlockDelayUs*, int *framePeriodUs*)

Create and add one or more CSDB ports to the session.

Parameters:

sessionHandle The handle to an existing session

resource The device and port(s) to add to the session.

bps The CSDB port speed in bits per second (12500 or 50000)

parity The parity (0 for even >0 for odd)

blockSize The number of bytes per block (including address and status bytes)

numberOfMessages The number of message blocks per frame

interByteDelayUs The delay between bytes within a block

interBlockDelayUs The delay between blocks within a frame

framePeriodUs The period at which cyclic frames are emitted

Returns:

the status code.

3.2.3.44 UeiDaqAPI int UeiDaqCreateDiagnosticChannel (SessionHandle *sessionHandle*, char * *resource*)

Create and add one or more channel to the diagnostic channel list associated with the session.

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

Returns:

The status code

3.2.3.45 UeiDaqAPI int UeiDaqCreateDIChannel (SessionHandle *sessionHandle*, char * *resource*)

Create and add one or more channel to the DI channel list associated with the session.

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

Returns:

The status code

3.2.3.46 UeiDaqAPI int UeiDaqCreateDIIndustrialChannel (SessionHandle *sessionHandle*, char * *resource*, double *lowThreshold*, double *highThreshold*, double *minPulseWidth*)

Create and add one or more channel to the channel list associated with the session. Each channel is associated with a digital input port that groups a predefined number of input lines. Industrial channels are only available on certain DIO devices such as the DNA-DIO-448. You can program program the levels at which the input line state change as well as configure a digital filter to eliminate glitches and spikes

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

lowThreshold the low hysteresis threshold

highThreshold the high hysteresis threshold

minPulseWidth the digital filter minimum pulse width in ms. Use 0.0 to disable digital input filter.

Returns:

the status code.

See also:

UeiDaqCreateAIChannel (p. 715), **UeiDaqCreateAOChannel** (p. 717), **UeiDaqCreateDIChannel** (p. 722), **UeiDaqCreateDOChannel** (p. 723), **UeiDaqCreateCIChannel** (p. 720), **UeiDaqCreateCOChannel** (p. 721)

3.2.3.47 UeiDaqAPI int UeiDaqCreateDMMChannel (SessionHandle *sessionHandle*, char * *resource*, f64 *range*, tUeiDMMMeasurementMode *measurementMode*)

Create and add one or more analog input channel to the channel list associated with the session. This will only work with DMM devices.

Example:

```
SessionHandle mySession;
UeiDaqCreateSession(&mySession);

UeiDaqCreateDMMChannel(mySession, "pdna://192.168.100.2/Dev1/Ai0", 10, UeiDMMModeDCVoltage);
```

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

range The input range. Input signals will need to be within -range/+range interval.

measurementMode The measurement mode configuration

Returns:

The status code.

See also:

UeiDaqCreateAIChannel (p. 715)

3.2.3.48 UeiDaqAPI int UeiDaqCreateDOChannel (SessionHandle *sessionHandle*, char * *resource*)

Create and add one or more channel to the DO channel list associated with the session.

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

Returns:

The status code

3.2.3.49 UeiDaqAPI int UeiDaqCreateDOIndustrialChannel (SessionHandle *sessionHandle*, char * *resource*, tUeiDOPWMMode *pwmMode*, uInt32 *pwmLengthUs*, uInt32 *pwmPeriodUs*, double *pwmDutyCycle*)

Create and add one or more channel to the channel list associated with the session. Each channel is associated with a digital output port that groups a predefined number of output lines. Industrial DO channels are available on certain devices such as the DNx-MF-101 and DNx-DIO-43x. low-to high and high to low transitions can be replaced by pwm providing soft start and soft stop.

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

pwmMode The PWM mode used to soft start and/or stop.

pwmLengthUs The soft start/stop pulse train length in micro-seconds.

pwmPeriodUs The soft start/stop or continuous pulse train period in micro-seconds.

pwmDutyCycle The continuous pulse train duty cycle

Returns:

The status code.

See also:

UeiDaqCreateDOChannel (p. 723)

3.2.3.50 UeiDaqAPI int UeiDaqCreateDOProtectedChannel (SessionHandle *sessionHandle*, char * *resource*, double *underCurrentLimit*, double *overCurrentLimit*, double *currentSampleRate*, int *autoRetry*, double *retryRate*)

Create and add one or more channel to the channel list associated with the session. Protected channels are available on certain devices such as the PowerDNA DIO-416. The amount of current flowing through each digital line is monitored at the given rate and must stay within the specified range, otherwise the device will automatically open the circuit acting as a breaker. You can specify whether the device should attempt to reestablish the circuit and how often it should try to do so.

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

underCurrentLimit The minimum amount of current allowed in Amps.

overCurrentLimit The maximum amount of current allowed in Amps.

currentSampleRate The current sampling rate. Determines how fast the breaker react after an over or under-current condition.

autoRetry Specifies whether the device will attempt to reestablish the circuit after an over or under-current condition

retryRate The number of retries per second.

Returns:

the status code.

See also:

UeiDaqCreateAIChannel (p. 715), **UeiDaqCreateAOChannel** (p. 717), **UeiDaqCreateDIChannel** (p. 722), **UeiDaqCreateDOChannel** (p. 723), **UeiDaqCreateCIChannel** (p. 720), **UeiDaqCreateCOChannel** (p. 721)

3.2.3.51 UeiDaqAPI int UeiDaqCreateHDLCPort (SessionHandle *sessionHandle*, char * *resource*, tUeiHDLCPortPhysical *physInterface*, uInt32 *bitsPerSecond*, tUeiHDLCPortEncoding *encoding*, tUeiHDLCPortCRCMode *crcMode*, tUeiHDLCPortClockSource *txClockSource*, tUeiHDLCPortClockSource *rxClockSource*)

Create and add one or more HDLC ports to the session.

Parameters:

sessionHandle The handle to an existing session
resource The device and port(s) to add to the session.
physInterface The physical interface: RS-232, RS-485 or RS-422
bitsPerSecond The number of bits transmitted per second
encoding The method used to encode bits on the physical interface
crcMode The method used to calculate each frame CRC
txClockSource The source of the transmitter clock
rxClockSource The source of the receiver clock

Returns:

the status code.

3.2.3.52 UeiDaqAPI int UeiDaqCreateI2CMasterPort (SessionHandle *sessionHandle*, char * *resource*, tUeiI2CPortSpeed *bitRate*, tUeiI2CTTLLevel *ttlLevel*, int *enableSecureShell*)

Create and add one or more port to the I2C master channel list associated with the session.

Parameters:

sessionHandle The handle to an existing session
resource The device and channel(s) to add to the list
bitRate The number of bits transmitted per second
ttlLevel The line voltage level
enableSecureShell Enable or disable use of secure shell functions for reading and writing

3.2.3.53 UeiDaqAPI int UeiDaqCreateI2CSlavePort (SessionHandle *sessionHandle*, char * *resource*, tUeiI2CTTLLevel *ttlLevel*, tUeiI2CSlaveAddressWidth *addressWidth*, int *address*)

Create and add one or more port to the I2C slave channel list associated with the session.

Parameters:

sessionHandle The handle to an existing session
resource The device and channel(s) to add to the list
ttlLevel The line voltage level
addressWidth The addressing mode used by the slave
address The I2C address of the slave

3.2.3.54 UeiDaqAPI int UeiDaqCreateIRIGDOTTLChannel (SessionHandle *sessionHandle*, char * *resource*, tUeiIRIGDOTTLSource *source0*, tUeiIRIGDOTTLSource *source1*, tUeiIRIGDOTTLSource *source2*, tUeiIRIGDOTTLSource *source3*)

Create and add one or more IRIG DO TTL channels to the session.

Parameters:

sessionHandle The handle to an existing session

resource The device and port(s) to add to the session.

source0 The source used to generate TTL pulses on output 0

source1 The source used to generate TTL pulses on output 1

source2 The source used to generate TTL pulses on output 2

source3 The source used to generate TTL pulses on output 3

Returns:

the status code.

3.2.3.55 UeiDaqAPI int UeiDaqCreateIRIGInputChannel (SessionHandle *sessionHandle*, char * *resource*, tUeiIRIGDecoderInputType *inputType*, tUeiIRIGTimeCodeFormat *timeCodeFormat*)

Create and add one or more IRIG input channels to the session.

Parameters:

sessionHandle The handle to an existing session

resource The device and port(s) to add to the session.

inputType The input where the timecode signal is connected

timeCodeFormat The time code format used by the input signal

Returns:

the status code.

3.2.3.56 UeiDaqAPI int UeiDaqCreateIRIGOutputChannel (SessionHandle *sessionHandle*, char * *resource*, tUeiIRIGTimeCodeFormat *timeCodeFormat*)

Create and add one or more IRIG output channels to the session.

Parameters:

sessionHandle The handle to an existing session

resource The device and port(s) to add to the session.

timeCodeFormat The time code format used for the generated signal

Returns:

the status code.

3.2.3.57 UeiDaqAPI int UeiDaqCreateIRIGTimeKeeperChannel (SessionHandle *sessionHandle*, char * *resource*, tUeiIRIGTimeKeeper1PPSSource *source*, int *autoFollow*)

Create and add one or more IRIG time keeper channels to the session.

Parameters:

sessionHandle The handle to an existing session

resource The device and port(s) to add to the session.

source The 1 PPS signal source

autoFollow Enable or disable auto follow

Returns:

the status code.

3.2.3.58 UeiDaqAPI int UeiDaqCreateLVDTChannel (SessionHandle *sessionHandle*, char * *resource*, f64 *min*, f64 *max*, double *sensorSensitivity*, tUeiLVDTWiringScheme *wiringScheme*, double *excitationVoltage*, double *excitationFrequency*, int *externalExcitation*)

Create and add one or more LVDT or RVDT input channel to the channel list associated with the session. This will only work with devices that can provide excitation waveform to the LVDT or RVDT sensor.

Example:

```
SessionHandle mySession;
UeiDaqCreateSession(&mySession);

// Configure channel 0 on device 1 to acquire position measured
// by a LVDT with a sensitivity of 24 mV/V/mm, powered by a 600Hz
// sine waveform with amplitude of 10.0V RMS .
// The gain of the device is adjusted to measure positions
// between -10.0mm and + 10.0mm
UeiDaqCreateLVDTChannel(mySession, "pwrdaq://Dev1/Ai0", -10.0, 10.0, 24,
                        UeiLVDTFiveWires, 10.0, 600.0, 0);
```

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

min The minimum value you expect to measure. The unit depends on the unit of the LVDT/RVDT sensitivity. (mV/V/mm will provide measurements in mm).

max The maximum value you expect to measure. The unit depends on the unit of the LVDT/RVDT sensitivity. (mV/V/mm will provide measurements in mm).

sensorSensitivity The sensitivity of the LVDT or RVDT. The unit of this parameter specifies the unit of the measurements.

wiringScheme Specifies whether the LVDT/RVDT is tied to the device using 4 or 5 wires

excitationVoltage The amplitude RMS of the excitation sine waveform to output to the sensor

excitationFrequency The frequency of the excitation sine waveform to output to the sensor

externalExcitation Specifies whther the LVDT is powered by the device or by an external source.

Returns:

The status code.

See also:

UeiDaqCreateAIChannel (p. 715)

3.2.3.59 UeiDaqAPI int UeiDaqCreateMIL1553Port (SessionHandle *sessionHandle*, char * *resource*, tUeiMIL1553PortCoupling *coupling*, tUeiMIL1553PortOpMode *portMode*)

Create and add one or more MIL-1553 ports to the session.

Parameters:

sessionHandle The handle to an existing session

resource The device and port(s) to add to the session.

coupling The way how the port is connected to MIL-1553 bus (tranformer, direct, etc.)

portMode The mode of operation (bus monitor, remote terminal, bus controller)

Returns:

the status code.

3.2.3.60 UeiDaqAPI int UeiDaqCreateMuxPort (SessionHandle *sessionHandle*, char * *resource*, int *breakBeforeMake*)

Create and add a MUX port to the session.

Parameters:

sessionHandle The handle to an existing session

resource The device and port to add to the session.

breakBeforeMake 1 to enable break before make, 0 otherwise

Returns:

the status code.

3.2.3.61 UeiDaqAPI int UeiDaqCreateQuadratureEncoderChannel (SessionHandle *sessionHandle*, char * *resource*, uInt32 *initialPosition*, tUeiQuadratureDecodingType *decodingType*, Int32 *enableZeroIndexing*, tUeiQuadratureZeroIndexPhase *zeroIndexPhase*)

Create and add one or more channel to the CI channel list associated with the session.

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

initialPosition The initial number of pulses when the session starts

decodingType The decoding type 1x, 2x or 4x

enableZeroIndexing Enable or disable restting the measurement when a zero index event is detected

zeroIndexPhase Specifies the states of A, B and Z inputs that will generate a zero index event

Returns:

The status code

3.2.3.62 UeiDaqAPI int UeiDaqCreateResistanceChannel (SessionHandle *sessionHandle*, char * *resource*, f64 *min*, f64 *max*, tUeiWiringScheme *wiring*, double *twoWiresLeadResistance*, tUeiAIChannelInputMode *mode*)

Create and add one or more resistance channel to the channel list associated with the session. In two wires mode, only one analog input channel per resistance is required. In four wires mode, two analog input channels per resistance are required effectively dividing the number of available channels on your device by two. Channels are paired consecutively: (0,1), (2,3), (4,5).... You only need to specify the first channel in the channel list. Read the DNA-STP-AIU manual to learn how to connect your resistance.

Example:

```
SessionHandle mySession;
UeiDaqCreateSession(&mySession);
UeiDaqCreateResistanceChannel(mySession, "pdna://192.168.100.2/Dev1/Ai0", 0.0, 1000.0,
                              UeiFourWires, 0.0);
```

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

min The minimum value you expect to measure.

max The maximum value you expect to measure.

wiring The wiring scheme used to connect the resistive sensor to the acquisition device.

twoWiresLeadResistance The leads resistance in two wires mode.

mode The input mode used to measure voltage from the sensor

Returns:

The status code.

See also:

UeiDaqCreateAIChannel (p. 715)

3.2.3.63 UeiDaqAPI int UeiDaqCreateRTDChannel (SessionHandle *sessionHandle*, char * *resource*, f64 *min*, f64 *max*, tUeiWiringScheme *wiring*, double *twoWiresLeadResistance*, tUeiRTDType *rtdType*, double *rtdNominalResistance*, tUeiTemperatureScale *tempScale*, tUeiAIChannelInputMode *mode*)

Create and add one or more RTD channel to the channel list associated with the session. In two wires mode, only one analog input channel per RTD is required. In four wires mode, two analog input channels per RTD are required effectively dividing the number of available channels on your device by two. Channels are paired consecutively: (0,1), (2,3), (3,4).... You only need to specify the first channel in the channel list. Read the DNA-STP-AIU manual to learn how to connect your RTD.

Example:

```
SessionHandle mySession;
UeiDaqCreateSession(&mySession);
UeiDaqCreateRTDChannel(mySession, "pdna://192.168.100.2/Dev1/Ai0", -100.0, 100.0, UeiFourWires, 0.0,
                       UeiRTDType3850, 100.0, UeiTemperatureScaleCelsius,
                       UeiAIChannelInputModeDifferential);
```

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

min The minimum value you expect to measure.

max The maximum value you expect to measure.

wiring The wiring scheme used to connect the RTD to the acquisition device.

twoWiresLeadResistance The leads resistance in two wires mdoe.

rtdType The type of RTD connected on the Analog input(s).

rtdNominalResistance The nominal resistance of the RTD at ice point (0 deg Celsius).

tempScale The temperature unit used to convert resistance to temperature.

mode The input mode used to measure voltage from the sensor

Returns:

The status code.

See also:

UeiDaqCreateAIChannel (p. 715)

3.2.3.64 UeiDaqAPI int UeiDaqCreateSerialPort (SessionHandle *sessionHandle*, char * *resource*, tUeiSerialPortMode *mode*, tUeiSerialPortSpeed *bitsPerSecond*, tUeiSerialPortDataBits *dataBits*, tUeiSerialPortParity *parity*, tUeiSerialPortStopBits *stopBits*, char * *termination*)

Create and add one or more serial ports to the session.

Parameters:

sessionHandle The handle to an existing session

resource The device and port(s) to add to the session.

mode The serial port mode: RS-232, RS-485 half ans full duplex

bitsPerSecond The number of bits transmitted per second over the serial link

dataBits The number of data bits describing each character

parity The parity scheme used for transmission error detection

stopBits The number of stop bits used to indicate the end of a data message

termination The read operation terminates when the termination string is read from the serial device

Returns:

the status code.

3.2.3.65 UeiDaqAPI int UeiDaqCreateSession (SessionHandle * *sessionHandle*)

Allocates and initializes resources for a new session.

Parameters:

sessionHandle The handle to the new session

Returns:

The status code

3.2.3.66 UeiDaqAPI int UeiDaqCreateSimulatedLVDTChannel (SessionHandle *sessionHandle*, char * *resource*, double *sensorSensitivity*, tUeiLVDTWiringScheme *wiringScheme*, double *excitationVoltage*, double *excitationFrequency*)

Create and add one or more simulated LVDT or RVDT output channel to the channel list associated with the session.

Example:

```
SessionHandle mySession;
UeiDaqCreateSession(&mySession);

// Configure channel 0 on device 1 to simulate position measured
// by a LVDT with a sensitivity of 24 mV/V/mm, powered by a 600Hz
// sine waveform with amplitude of 10.0V RMS.
UeiDaqCreateSimulatedLVDTChannel(mySession, "pwrdaq://Dev1/Ai0", 24,
                                 UeiLVDTFiveWires, 10.0, 600.0);
```

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

sensorSensitivity The sensitivity of the LVDT or RVDT. The unit of this parameter specifies the unit of the measurements.

wiringScheme Specifies whether the LVDT/RVDT is tied to the device using 4 or 5 wires

excitationVoltage The amplitude RMS of the excitation sine waveform to output to the sensor

excitationFrequency The frequency of the excitation sine waveform to output to the sensor

Returns:

The status code.

See also:

UeiDaqCreateAOChannel (p. 717)

3.2.3.67 UeiDaqAPI int UeiDaqCreateSimulatedRTDChannel (SessionHandle *sessionHandle*, char * *resource*, tUeiRTDType *rtdType*, double *rtdNominalResistance*, tUeiTemperatureScale *tempScale*)

Create and add one or more simulated RTD channel to the channel list associated with the session.

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

rtdType The type of RTD connected on the Analog input(s).

rtdNominalResistance The nominal resistance of the RTD at ice point (0 deg Celsius).

tempScale The temperature unit used to convert resistance to temperature.

Returns:

The status code.

3.2.3.68 UeiDaqAPI int UeiDaqCreateSimulatedSynchroResolverChannel (SessionHandle *sessionHandle*, char * *resource*, tUeiSynchroResolverMode *mode*, double *excitationVoltage*, double *excitationFrequency*, int *externalExcitation*)

Create and add one or more simulated synchro or simulated resolver channel to the channel list associated with the session.

Example:

```
SessionHandle mySession;
UeiDaqCreateSession(&mySession);

// Configure channel 0 on device 1 to simulate position measurement
// returned by a synchro powered by a 600Hz sine waveform with
// amplitude of 10.0 VRMS.
UeiDaqCreateSimulatedSynchroResolverChannel(mySession, "pwrdaq://Dev1/Ai0",
                                            UeiSynchroMode, 10.0, 600.0, false);
```

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

mode Specifies whether the sensor simulated is a synchro or a resolver

excitationVoltage The amplitude RMS of the excitation sine waveform to output to the sensor

excitationFrequency The frequency of the excitation sine waveform to output to the sensor

externalExcitation Specifies whther the sensor is powered by the device or by an external source.

Returns:

The status code.

See also:

CreateAIChannel

3.2.3.69 UeiDaqAPI int UeiDaqCreateSimulatedTCChannel (SessionHandle *sessionHandle*, char * *resource*, tUeiThermocoupleType *tcType*, tUeiTemperatureScale *tempScale*, int *enableCjc*)

Create and add one or more simulated thermocouple channel to the session.

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

tcType The type of thermocouple to simulate.

tempScale The temperature unit used to convert temperature to volts.

enableCjc The cold junction compensation state.

Returns:

The status code

3.2.3.70 UeiDaqAPI int UeiDaqCreateSSIMasterPort (SessionHandle *sessionHandle*, char * *resource*, unsigned int *bps*, unsigned int *wordSize*, int *enableClock*, int *enableTermination*, double *pauseTime*, double *transferTimeout*, double *bitUpdateTime*)

Create and add one or more port to the SSI channel list associated with the session.

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

bps The speed of the clock

wordSize The number of bits per word (3 to 32)

enableClock Enable or disable clock output

enableTermination Enable or disable termination resistor

pauseTime Specifies the time delay between two consecutive clock sequences from the master

transferTimeout Specifies the minimum time required by the slave to realise that the data transmission is complete

bitUpdateTime Specifies the maximum bit update time

Returns:

the status code.

3.2.3.71 UeiDaqAPI int UeiDaqCreateSSISlavePort (SessionHandle *sessionHandle*, char * *resource*, unsigned int *bps*, unsigned int *wordSize*, int *enableTransmit*, int *enableTermination*, double *pauseTime*, double *transferTimeout*, double *bitUpdateTime*)

Create and add one or more port to the SSI channel list associated with the session.

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

bps The speed of the clock

wordSize The number of bits per word (3 to 32)

enableTransmit Enable or disable data output

enableTermination Enable or disable termination resistor

pauseTime Specifies the time delay between two consecutive clock sequences from the master

transferTimeout Specifies the minimum time required by the slave to realise that the data transmission is complete

bitUpdateTime Specifies the maximum bit update time

Returns:

the status code.

3.2.3.72 UeiDaqAPI int UeiDaqCreateSync1PPSPort (SessionHandle *sessionHandle*, char * *resource*, tUeiSync1PPSMode *mode*, tUeiSync1PPSSource *source*, double *EMOutputRate*)

The 1PPS synchronization port uses an ADPLL and an event module to produce a clock at an arbitrary rate Multiple racks can easily be synchronized when they use the same 1PPS synchronization clock The ADPLL+event module on each rack will produce synchronized scan clocks

The source of the 1PPS sync clock can be internal, external or generated from NTP The 1PPS clock is routed via one of the four internal sync lines to the ADPLL (adaptive digital PLL) The ADPLL locks on the 1PPS and outputs its own 1PPS that is an average of the original 1PPS clock The ADPLL can maintain its 1PPS output even if the original 1PPS clock gets disconnected The ADPLL 1PPS output is routed via one of the four sync lines to the event module The ADPLL 1PPS output can also be routed to the sync connector for sharing with other racks/cubes The event module produces a user selectable number of pulses upon every 1PPS pulse coming from the ADPLL. The event module clock output is routed to the Input layers via one of the remaining sync lines

Parameters:

sessionHandle The handle to an existing session

resource The IO module IP address.

mode The mode used to obtain the reference 1PPS (existing 1PPS clock, NTP or 1588)

source The source of the 1PPS refenrece clock

EMOutputRate The rate of the resulting clock (synchronized with the 1PPS reference)

Returns:

the status code.

3.2.3.73 UeiDaqAPI int UeiDaqCreateSynchroResolverChannel (SessionHandle *sessionHandle*, char * *resource*, tUeiSynchroResolverMode *mode*, double *excitationVoltage*, double *excitationFrequency*, int *externalExcitation*)

Create and add one or more Synchro or Resolver channel to the channel list associated with the session. This will only work with devices that can provide excitation waveform to the Synchro or Resolver sensor.

Example:

```
SessionHandle mySession;
UeiDaqCreateSession(&mySession);

// Configure channel 0 on device 1 to acquire position measured
// by a synchro powered by a 600Hz sine waveform with
// amplitude of 10.0 VRMS.
UeiDaqCreateSynchroResolverChannel(mySession, "pwrdaq://Dev1/Ai0",
                                   UeiSynchroMode, 10.0, 600.0, false);
```

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

mode Specifies whether the input sensor is a synchro or a resolver

excitationVoltage The amplitude RMS of the excitation sine waveform to output to the sensor

excitationFrequency The frequency of the excitation sine waveform to output to the sensor

externalExcitation Specifies whther the sensor is powered by the device or by an external source.

Returns:

The status code.

See also:

CreateAIChannel

3.2.3.74 UeiDaqAPI int UeiDaqCreateTCChannel (SessionHandle *sessionHandle*, char * *resource*, f64 *min*, f64 *max*, tUeiThermocoupleType *tcType*, tUeiTemperatureScale *tempScale*, tUeiColdJunctionCompensationType *cjcType*, f64 *cjcConstant*, char * *cjcResource*, tUeiAIChannelInputMode *mode*)

Create and add one or more thermocouple channel to the thermocouple channel list associated with the session.

Example:

```
SessionHandle mySession;
UeiDaqCreateSession(&mySession);
UeiDaqCreateTCChannel(mySession, "pwrdaq://Dev1/Ai0", -100.0, 100.0, UeiThermocoupleTypeJ,
                          UeiTemperatureScaleCelsius, UeiCJCTypeConstant, 25.0, "",
                          UeiAIChannelInputModeDifferential);
```

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

min The minimum value you expect to measure.

max The maximum value you expect to measure.

tcType The type of thermocouple connected on the Analog input(s).

tempScale The temperature unit used to convert volts to temperature.

cjcType The cold junction compensation type.

cjcConstant The cold junction compensation constant, this parameter is only used if cjcType is set to UeiCJCTypeConstant.

cjcResource The resource string of the channel on which the cold junction compensation sensor is connected, this parameter is only used if cjcType is set to UeiCJCTypeResource.

mode The input mode used to measure voltage from the thermocouple

Returns:

The status code.

See also:

UeiDaqCreateAIChannel (p. 715)

3.2.3.75 UeiDaqAPI int UeiDaqCreateVRChannel (SessionHandle *sessionHandle*, char * *resource*, tUeiVRMode *mode*)

Create and add one or more channel to the VR channel list associated with the session.

Parameters:

sessionHandle The handle to an existing session

resource The device and channel(s) to add to the list.

mode The mode used to decode/count the VR sensor signal

Returns:

The status code

3.2.3.76 UeiDaqAPI int UeiDaqEnableAccelLowPassfilter (ChannelHandle *channelHandle*, int *enabled*)

Enable or disable the low-pass filter.

Parameters:

channelHandle The handle to an existing accelerometer channel

enabled The low-pass filter state

Returns:

The status code.

See also:

UeiDaqIsAccelLowPassFilterEnabled (p. 848)

3.2.3.77 UeiDaqAPI int UeiDaqEnableAIChannelAutoZero (ChannelHandle *channelHandle*, int *enable*)

Some analog input devices come with a special channel to measure ground offset. auto-zero substract the ground voltage measurement from the input voltage measurement.

Parameters:

channelHandle The handle to an existing channel

enable the new auto-zero state

Returns:

The status code

3.2.3.78 UeiDaqAPI int UeiDaqEnableAIChannelBias (ChannelHandle *channelHandle*, int *enable*)

Bias resistors connects the negative input to ground and positive input to a voltage source. This is useful to measure temperature from un-grounded thremocouple. Disable bias to measure from grounded thermocouples

Parameters:

channelHandle The handle to an existing channel

enable the bias resistor state

Returns:

The status code

3.2.3.79 UeiDaqAPI int UeiDaqEnableAIChannelOpenCircuitTest (ChannelHandle *channelHandle*, int *enable*)

Enable or disable open circuit detection

Parameters:

channelHandle The handle to an existing channel

enable true to enable open circuit detection

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.80 UeiDaqAPI int UeiDaqEnableAIVExOffsetNulling (ChannelHandle *channelHandle*, int *enableNulling*)

Enables or Disables offset nulling circuitry With Offset nulling enabled, a nulling circuit adds an adjustable DC voltage to the output of the amplifier making sure that the bridge output is 0V when no strain is applied.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

enableNulling 1 to enable offset nulling, 0 to disable it

Returns:

The status code.

See also:

UeiDaqIsAIVExOffsetNullingEnabled (p. 850)

3.2.3.81 UeiDaqAPI int UeiDaqEnableAIVExShuntCalibration (ChannelHandle *channelHandle*, int *engageShuntCal*)

Connect or disconnect the shunt calibration resistor.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

engageShuntCal true to enable the shunt resistor, false to disable it

Returns:

The status code.

See also:

UeiDaqIsAIVExShuntCalibrationEnabled (p. 850)

3.2.3.82 UeiDaqAPI int UeiDaqEnableAOChannelDefaultValue (ChannelHandle *channelHandle*, int *enableDefaultValue*)

Enables default value. The output will be set to a default value when session stops.

Parameters:

channelHandle The handle to an existing channel

enableDefaultValue true to turn stop value on, false otherwise

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.83 UeiDaqAPI int UeiDaqEnableAOProtectedCircuitBreaker (ChannelHandle *channelHandle*, int *index*, int *enable*)

Enable or disable circuit breaker on this channel. when enabled a circuit breaker monitors up to two measurement channels and opens the circuit if any of the measurements goes out of pre-set limits.

Parameters:

channelHandle The handle to an existing channel

index The 0 based index of the circuit breaker

enable 1 to turn-on protection, 0 to turn it off

Returns:

The status code

3.2.3.84 UeiDaqAPI int UeiDaqEnableARINCInputLabelFilter (ChannelHandle *channelHandle*, int *enableLabelFilter*)

Enables label filtering.

Parameters:

channelHandle The handle to an existing ARINC-429 input channel

enableLabelFilter 1 to turn label filtering on, 0 otherwise

Returns:

The status code.

See also:

UeiDaqIsARINCInputLabelFilterEnabled (p. 851)

3.2.3.85 UeiDaqAPI int UeiDaqEnableARINCInputSDIFilter (ChannelHandle *channelHandle*, int *enableSDIFilter*)

Specifies whether the SDI filter is turned on or off.

Parameters:

channelHandle The handle to an existing ARINC-429 input channel

enableSDIFilter true to turn SDI filtering on, false otherwise

Returns:

The status code.

See also:

UeiDaqIsARINCInputSDIFilterEnabled (p. 852)

3.2.3.86 UeiDaqAPI int UeiDaqEnableARINCInputSlowSlewRate (ChannelHandle *channelHandle*, int *enableSlowSlewRate*)

Enables slow slew rate.

Parameters:

channelHandle The handle to an existing ARINC-429 input channel

enableSlowSlewRate 1 to turn slow slew rate on, 0 otherwise

Returns:

The status code.

See also:

UeiDaqIsARINCInputSlowSlewRateEnabled (p. 852)

3.2.3.87 UeiDaqAPI int UeiDaqEnableARINCInputTimestamping (ChannelHandle *channelHandle*, int *enableTimestamping*)

Specifies whether each received frame is timestamped.

Parameters:

channelHandle The handle to an existing ARINC-429 input channel

enableTimestamping 1 to turn timestamping on, 0 otherwise

Returns:

The status code.

See also:

UeiDaqIsARINCInputTimestampingEnabled (p. 852)

3.2.3.88 UeiDaqAPI int UeiDaqEnableARINCOutputLoopback (ChannelHandle *channelHandle*, int *enableLoopback*)

Enables loopback. When enabled you can read back packet sent to this port on a dedicated loopback port.

Parameters:

channelHandle The handle to an existing ARINC-429 output channel

enableLoopback 1 to enable loopback, 0 to disable it.

Returns:

The status code.

See also:

UeiDaqIsARINCOutputLoopbackEnabled (p. 853)

3.2.3.89 UeiDaqAPI int UeiDaqEnableARINCOutputScheduler (ChannelHandle *channelHandle*, int *enableScheduler*)

Enables scheduling.

Parameters:

channelHandle The handle to an existing ARINC-429 output channel

enableScheduler 1 to turn scheduling on, 0 otherwise

Returns:

The status code.

See also:

UeiDaqIsARINCOutputSchedulerEnabled (p. 853)

3.2.3.90 UeiDaqAPI int UeiDaqEnableARINCOutputSlowSlewRate (ChannelHandle *channelHandle*, int *enableSlowSlewRate*)

Enables slow slew rate.

Parameters:

channelHandle The handle to an existing ARINC-429 output channel

enableSlowSlewRate 1 to turn slow slew rate on, 0 otherwise

Returns:

The status code.

See also:

UeiDaqIsSlowARINCOutputSlewRateEnabled

3.2.3.91 UeiDaqAPI int UeiDaqEnableARINCScheduler (SessionHandle *sessionHandle*, Int32 *port*, int *enable*)

Enable or disable scheduler

Parameters:

sessionHandle The handle to an existing session

port the port to enable/disable

enable true to enable, false to disable scheduler

Returns:

The status code

3.2.3.92 UeiDaqAPI int UeiDaqEnableCANPortWarningAndErrorLogging (ChannelHandle *channelHandle*, int *enable*)

When logging is enabled, bus warnings and errors are sent in the data stream. Use the Type field of the CANDataFrame structure to determine whether received frames are data or error frames.

Parameters:

channelHandle The handle to an existing CAN port

enable 1 to enable error logging, 0 to disable it

Returns:

The status code.

See also:

UeiDaqIsCANPortWarningAndErrorLoggingEnabled (p. 854)

3.2.3.93 UeiDaqAPI int UeiDaqEnableDataStreamBlocking (DataStreamHandle *dataStreamHandle*, Int32 *enable*)

When enabled, read operations will block until the requested amount of data is received from the device. when disabled, read operations return immediately with the amount of data available.

Parameters:

dataStreamHandle The handle to an existing data stream

blocking the blocking setting value.

Returns:

The status code

3.2.3.94 UeiDaqAPI int UeiDaqEnableDIIndustrialACMode (ChannelHandle *channelHandle*, int *line*, int *acMode*)

Enable or disable AC mode In AC mode, the RMS voltage measured at each input is compared with lowa nd high thresholds to determine the state of the input

Parameters:

channelHandle The handle to an existing industrial DI channel

line The input line to configure

acMode true to enable AC mode, false otherwise

Returns:

The status code.

3.2.3.95 UeiDaqAPI int UeiDaqEnableDOChannelDefaultValue (ChannelHandle *channelHandle*, int *enableDefaultValue*)

Enables default value. The output will be set to a default value when session stops.

Parameters:

channelHandle The handle to an existing channel

enableDefaultValue true to turn stop value on, false otherwise

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.96 UeiDaqAPI int UeiDaqEnableDOProtectedCircuitBreaker (ChannelHandle *channelHandle*, int *line*, int *enable*)

Enable or disable circuit breaker on this output line. when enabled a circuit breaker monitors the current flowing through the line and opens the circuit if the current goes out of pre-set limits.

Parameters:

channelHandle The handle to an existing channel

line The output line to configure within the port

enable 1 to turn-on protection, 0 to turn it off

Returns:

The status code

3.2.3.97 UeiDaqAPI int UeiDaqEnableI2CMasterSecureShell (ChannelHandle *channelHandle*, int *enable*)

Set the I2C secure shell setting

Parameters:

channelHandle The handle to an I2C master port

enable The new I2C secure shell setting

Returns:

The status code

3.2.3.98 UeiDaqAPI int UeiDaqEnableI2CMasterTerminationResistor (ChannelHandle *channelHandle*, int *enable*)

Set the I2C termination resistor setting

Parameters:

channelHandle The handle to an I2C master port
resistance The new I2C termination resistor setting

Returns:

The status code

3.2.3.99 UeiDaqAPI int UeiDaqEnableI2CMultiMaster (ChannelHandle *channelHandle*, int *enable*)

Set the I2C multi-master setting

Parameters:

channelHandle The handle to an I2C master port
enable The new I2C multi-master setting

Returns:

The status code

3.2.3.100 UeiDaqAPI int UeiDaqEnableI2CSlaveAddressClockStretching (ChannelHandle *channelHandle*, int *enable*)

Enable clock stretching during address cycle.

Parameters:

channelHandle The handle to an I2C master port
enable the new clock stretching during address cycle state

Returns:

The status code

3.2.3.101 UeiDaqAPI int UeiDaqEnableI2CSlaveBusMonitor (ChannelHandle *channelHandle*, int *enable*)

Set the I2C bus monitor setting

Parameters:

channelHandle The handle to an I2C slave port
enable The new I2C bus monitor setting

Returns:

The status code

3.2.3.102 UeiDaqAPI int UeiDaqEnableI2CSlaveBusMonitorAck (ChannelHandle *channelHandle*, int *enable*)

Set the I2C bus monitor ACK setting

Parameters:

channelHandle The handle to an I2C slave port

enable The new I2C bus monitor ACK setting

Returns:

The status code

3.2.3.103 UeiDaqAPI int UeiDaqEnableI2CSlaveReceiveClockStretching (ChannelHandle *channelHandle*, int *enable*)

Enable clock stretching during receive cycle.

Parameters:

channelHandle The handle to an I2C master port

enable the new clock stretching during receive cycle state

Returns:

The status code

3.2.3.104 UeiDaqAPI int UeiDaqEnableI2CSlaveTransmitClockStretching (ChannelHandle *channelHandle*, int *enable*)

Enable clock stretching during transmit cycle.

Parameters:

channelHandle The handle to an I2C master port

enable the new clock stretching during transmit cycle state

Returns:

The status code

3.2.3.105 UeiDaqAPI int UeiDaqEnableIRIGTimeKeeperAutoFollow (ChannelHandle *channelHandle*, int *enableAutoFollow*)

If selected external 1PPS source does not deliver pulses (because of a break in timecode transmission, for example). Timekeeper can switch to internal timebase when externally derived one is not available

Parameters:

channelHandle The handle to an existing MIL-1553 channel

enableAutoFollow true to turn auto-follow on, false otherwise

Returns:

The status code

See also:

UeiDaqIsIRIGAutoFollowEnabled

3.2.3.106 UeiDaqAPI int UeiDaqEnableIRIGTimeKeeperInvalidDay (ChannelHandle *channelHandle*, int *enableInvalidDay*)

Select whether days information is invalid in the input timecode

Parameters:

channelHandle The handle to an existing IRIG channel

enableInvalidDay true to specify that day is invalid, false otherwise

Returns:

The status code

See also:

IsInvalidIRIGDayEnabled

3.2.3.107 UeiDaqAPI int UeiDaqEnableIRIGTimeKeeperInvalidMinute (ChannelHandle *channelHandle*, int *enableInvalidMinute*)

Select whether minutes information is invalid in the input timecode

Parameters:

channelHandle The handle to an existing IRIG channel

enableInvalidMinute true to specify that minute is invalid, false otherwise

Returns:

The status code

See also:

UeiDaqIsIRIGInvalidMinuteEnabled

3.2.3.108 UeiDaqAPI int UeiDaqEnableIRIGTimeKeeperInvalidSecond (ChannelHandle *channelHandle*, int *enableInvalidSecond*)

Select whether seconds information is invalid in the input timecode

Parameters:

channelHandle The handle to an existing IRIG channel

enableInvalidSecond true to specify that second is invalid, false otherwise

Returns:

The status code

See also:

UeiDaqIsIRIGInvalidSecondEnabled

3.2.3.109 UeiDaqAPI int UeiDaqEnableIRIGTimeKeeperNominalValue (ChannelHandle *channelHandle*, int *enableNominalValue*)

Select whether to use nominal period (i.e. 100e6 pulses of 100MHz base clock) or the period measured by timekeeper (it measures and averages number of base clock cycles between externally derived 1PPS pulses when they are valid).

Parameters:

channelHandle The handle to an existing MIL-1553 channel

enableNominalValue true to turn nominal value on, false otherwise

Returns:

The status code

See also:

UeiDaqIsIRIGNominalValueEnabled

3.2.3.110 UeiDaqAPI int UeiDaqEnableIRIGTimeKeeperSBS (ChannelHandle *channelHandle*, int *enableSBS*)

Select whether to use SBS timecode section for TimeKeeper hour/min decoding (if BCD data in the incoming timecode is corrupted)

Parameters:

channelHandle The handle to an existing IRIG input channel

enableSBS true to use SBS data instead of BCD data, false otherwise

Returns:

The status code

See also:

UeiDaqIsIRIGSBSEnabled

3.2.3.111 UeiDaqAPI int UeiDaqEnableIRIGTimeKeeperSubPPS (ChannelHandle *channelHandle*, int *enableSubPPS*)

Select whether external timebase is slower than 1PPS or is not derived from the timecode

Parameters:

channelHandle The handle to an existing MIL-1553 channel

enableSubPPS true to turn sub PPS on, false otherwise

Returns:

The status code

See also:

UeiDaqIsIRIGSubPPSEnabled

3.2.3.112 UeiDaqAPI int UeiDaqEnableLVDTExternalExcitation (ChannelHandle *channelHandle*, int *enabled*)

Enable or disable the external excitation.

Parameters:

channelHandle The handle to an existing LVDT channel

enabled The new external excitation state

Returns:

The status code.

See also:

UeiDaqIsLVDTExternalExcitationEnabled (p. 860)

3.2.3.113 UeiDaqAPI int UeiDaqEnableMIL1553Filter (ChannelHandle *channelHandle*, int *enableFilter*)

Enables filtering.

Parameters:

channelHandle The handle to an existing MIL-1553 input channel

enableFilter 1 to turn label filtering on, 0 otherwise

Returns:

The status code.

See also:

UeiDaqIsMIL1553FilterEnabled (p. 861)

3.2.3.114 UeiDaqAPI int UeiDaqEnableMIL1553Scheduler (ChannelHandle *channelHandle*, int *enableScheduler*)

Enables scheduling.

Parameters:

channelHandle The handle to an existing MIL-1553 channel

enableScheduler 1 to turn scheduling on, 0 otherwise

Returns:

The status code.

See also:

UeiDaqIsMIL1553SchedulerEnabled (p. 861)

3.2.3.115 UeiDaqAPI int UeiDaqEnableMIL1553Timestamping (ChannelHandle *channelHandle*, int *enableTimestamping*)

Specifies whether each received frame is timestamped.

Parameters:

channelHandle The handle to an existing MIL-1553 channel

enableTimestamping 1 to turn timestamping on, 0 otherwise

Returns:

The status code.

See also:

UeiDaqIsMIL1553TimestampingEnabled (p. 861)

3.2.3.116 UeiDaqAPI int UeiDaqEnableMuxBreakBeforeMake (ChannelHandle *channelHandle*, int *enable*)

Set the port break before make setting

Parameters:

channelHandle The handle to a MUX port

enable true to enable, false otherwise

Returns:

The status code

3.2.3.117 UeiDaqAPI int UeiDaqEnableMuxSyncInput (ChannelHandle *channelHandle*, int *enable*)

Set the port synchronization input setting A MUX device can wait for a pulse on its synchronization input before configuring its relays

Parameters:

channelHandle The handle to a MUX port

enable true to enable, false otherwise

Returns:

The status code

3.2.3.118 UeiDaqAPI int UeiDaqEnableMuxSyncInputEdgeMode (ChannelHandle *channelHandle*, int *enable*)

Set the port synchronization input edge/level setting The sync input can work in level or edge mode

Parameters:

channelHandle The handle to a MUX port

enable true to enable, false otherwise

Returns:

The status code

3.2.3.119 UeiDaqAPI int UeiDaqEnableSerialPortErrorReporting (ChannelHandle *channelHandle*, int *enabled*)

Enable or disable reproting of framing and parity errors.

Parameters:

channelHandle The handle to an existing serial port

enabled The new error reporting state

Returns:

The status code.

See also:

UeiDaqIsSerialPortErrorReportingEnabled (p. 862)

3.2.3.120 UeiDaqAPI int UeiDaqEnableSerialPortHDEchoSuppression (ChannelHandle *channelHandle*, int *enabled*)

Enable or disable the half-duplex echo suppression.

Parameters:

channelHandle The handle to an existing serial port

enabled The new half-duplex echo suppression state

Returns:

The status code.

See also:

UeiDaqIsSerialPortHDEchoSuppressionEnabled (p. 863)

3.2.3.121 UeiDaqAPI int UeiDaqEnableSerialPortOnTheFlyParityBit (ChannelHandle *channelHandle*, int *enabled*)

Enable or disable on the fly parity bit update as part of the data stream (using uInt16 data instead of uInt8)

Parameters:

channelHandle The handle to an existing serial port

enabled The new on the fly parity bit state

Returns:

The status code.

See also:

UeiDaqIsSerialPortOnTheFlyParityBitEnabled (p. 863)

3.2.3.122 UeiDaqAPI int UeiDaqEnableSerialPortRxTerminationResistor (ChannelHandle *channelHandle*, int *enabled*)

Enable or disable the RX termination resistor.

Parameters:

channelHandle The handle to an existing serial port

enabled The new RX termination resistor state

Returns:

The status code.

See also:

UeiDaqIsSerialPortRxTerminationResistorEnabled (p. 863)

3.2.3.123 UeiDaqAPI int UeiDaqEnableSerialPortTxAutoDisable (ChannelHandle *channelHandle*, int *enabled*)

Specifies whether transmitter should automatically disable itself whenever the TX FIFO is empty (only used in RS-485 full duplex mode) This allows multiple RS-485 devices to share the same bus

Parameters:

channelHandle The handle to an existing serial port

enabled The new TX auto-disable state

Returns:

The status code.

See also:

UeiDaqIsSerialPortTxAutoDisableEnabled (p. 864)

3.2.3.124 UeiDaqAPI int UeiDaqEnableSerialPortTxTerminationResistor (ChannelHandle *channelHandle*, int *enabled*)

Enable or disable the TX termination resistor.

Parameters:

channelHandle The handle to an existing serial port

enabled The new TX termination resistor state

Returns:

The status code.

See also:

UeiDaqIsSerialPortTxTerminationResistorEnabled (p. 864)

3.2.3.125 UeiDaqAPI int UeiDaqEnableSessionAutoReStart (SessionHandle *sessionHandle*, int *enable*)

Enabling auto-restart will automatically stop/start a running session once it is detected that the device was reset (uually because of a power-cycle).

Parameters:

sessionHandle The handle to an existing session

enable true to auto-restart, false otherwise

Returns:

The status code

3.2.3.126 UeiDaqAPI int UeiDaqEnableSimulatedSynchroResolverExternalExcitation (ChannelHandle *channelHandle*, int *enabled*)

Enable or disable the external excitation.

Parameters:

channelHandle The handle to an existing simulated synchro/resolver channel

enabled The new external excitation state

Returns:

The status code.

See also:

UeiDaqIsSimulatedSynchroResolverExternalExcitationEnabled (p. 865)

3.2.3.127 UeiDaqAPI int UeiDaqEnableSynchroResolverExternalExcitation (ChannelHandle *channelHandle*, int *enabled*)

Enable or disable the external excitation.

Parameters:

channelHandle The handle to an existing synchro/resolver channel

enabled The new external excitation state

Returns:

The status code.

See also:

UeiDaqIsSynchroResolverExternalExcitationEnabled (p. 865)

3.2.3.128 UeiDaqAPI int UeiDaqEnumDevice (char * *deviceResourceName*, DeviceHandle * *deviceHandle*)

Enumerate the devices handled by the specified driver

Parameters:

deviceResourceName The name of the driver

deviceHandle The handle the device enumerated

Returns:

The status code

See also:

UeiDaqEnumDriver (p. 754)

3.2.3.129 UeiDaqAPI int UeiDaqEnumDriver (int *index*, char * *driverName*)

Enumerate the installed drivers

Parameters:

index The index of the driver to enumerate

driverName The name of the driver enumerated

Returns:

The status code

See also:

UeiDaqEnumDevice (p. 753)

3.2.3.130 UeiDaqAPI int UeiDaqFireTrigger (TriggerHandle *triggerHandle*)

Manually send a trigger event. This method can only be called When trigger source is set to UeiTriggerSourceSignal and the trigger signal is set to a software assertable signal.

Parameters:

triggerHandle The handle to an existing trigger object

Returns:

The status code

See also:

UeiDaqGetStartTriggerHandle (p. 833) **UeiDaqGetStopTriggerHandle** (p. 833)

3.2.3.131 UeiDaqAPI int UeiDaqFlushMessages (SessionHandle *sessionHandle*, Int32 *port*, tUeiFlushAction *action*)

Flush transmit and or receive buffer(s) associated with the specified port

Parameters:

sessionHandle The handle to an existing session

port the port to flush

action the action to be taken while flushing the buffer(s)

Returns:

The status code

3.2.3.132 UeiDaqAPI int UeiDaqGetAccelCouplingType (ChannelHandle *channelHandle*, tUeiCoupling * *pCoupling*)

Get the channel coupling type, AC or DC

Parameters:

channelHandle The handle to an existing accelerometer channel

pCoupling The coupling type

Returns:

The status code.

See also:

UeiDaqSetAccelCouplingType (p. 887)

3.2.3.133 UeiDaqAPI int UeiDaqGetAccelExcitationCurrent (ChannelHandle *channelHandle*, double * *pExcCurrent*)

Get the excitation current in mAmps

Parameters:

channelHandle The handle to an existing accelerometer channel

pExcCurrent The excitation current

Returns:

The status code.

See also:

UeiDaqSetAccelExcitationCurrent (p. 887)

3.2.3.134 UeiDaqAPI int UeiDaqGetAccelHighAlarmStatus (ChannelHandle *channelHandle*, int * *pHighStatus*)

Return the high alarm status which indicates whether the connection to the sensor is open. It also is visible on the terminal block as a steady red LED.

Parameters:

channelHandle The handle to an existing accelerometer channel

pHighStatus the high alarm status

See also:

UeiDaqSetAccelHighExcitationComparator (p. 887)

3.2.3.135 UeiDaqAPI int UeiDaqGetAccelHighExcitationComparator (ChannelHandle *channelHandle*, double * *pHighComparator*)

The excitation current can be measured back and trigger an alarm when it is out of range (due to an open connection or a short). It is measured as a voltage which depends on the sensor resistance and the programmed excitation current. The alarm state is made visible with the LED color on the terminal block Green means that the measured excitation is within range, red means that the connection with the sensor is open and blinking red means that the connection with the sensor is shorted.

Parameters:

channelHandle The handle to an existing accelerometer channel

pHighComparator the high excitation comparator voltage

Returns:

The status code.

See also:

UeiDaqSetAccelHighExcitationComparator (p. 887)

3.2.3.136 UeiDaqAPI int UeiDaqGetAccelLowAlarmStatus (ChannelHandle *channelHandle*, int * *pLowStatus*)

Return the low alarm status which indicates whether the connection to the sensor is shorted. It also is visible on the terminal block as a red blinking LED.

Parameters:

channelHandle The handle to an existing accelerometer channel

pLowStatus the low alarm status

See also:

UeiDaqSetAccelLowExcitationComparator (p. 888)

3.2.3.137 UeiDaqAPI int UeiDaqGetAccelLowExcitationComparator (ChannelHandle *channelHandle*, double * *pLowComparator*)

The excitation current can be measured back and trigger an alarm when it is out of range (due to an open connection or a short). It is measured as a voltage which depends on the sensor resistance and the programmed excitation current. The alarm state is made visible with the LED color on the terminal block Green means that the measured excitation is within range, red means that the connection with the sensor is open and blinking red means that the connection with the sensor is shorted.

Parameters:

channelHandle The handle to an existing accelerometer channel

pLowComparator the low excitation comparator voltage

Returns:

The status code.

See also:

UeiDaqSetAccelLowExcitationComparator (p. 888)

3.2.3.138 UeiDaqAPI int UeiDaqGetAccelSensorSensitivity (ChannelHandle *channelHandle*, double * *pSensitivity*)

Get the sensor sensitivity, it usually is in mVolts/g Independently of the unit, the voltage read will be converted to mV and divided by the sensitivity

Parameters:

channelHandle The handle to an existing accelerometer channel

pSensitivity The current sensitivity

Returns:

The status code.

See also:

UeiDaqSetAccelSensorSensitivity (p. 888)

3.2.3.139 UeiDaqAPI int UeiDaqGetAIChannelGain (ChannelHandle *channelHandle*, f64 * *gain*)

Get the gain that best fits the requested input range

Parameters:

channelHandle The handle to an existing channel

gain The current gain

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.140 UeiDaqAPI int UeiDaqGetAIChannelInputMode (ChannelHandle *channelHandle*, tUeiAIChannelInputMode * *mode*)

Get the input mode of the channel, possible values are "differential" or "single-ended".

Parameters:

channelHandle The handle to an existing channel

mode The returned input mode.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.141 UeiDaqAPI int UeiDaqGetAIChannelMaximum (ChannelHandle *channelHandle*, f64 * *maximum*)

Get the maximum expected value for the signal connected to the channel.

Parameters:

channelHandle The handle to an existing channel

maximum The returned maximum value.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.142 UeiDaqAPI int UeiDaqGetAIChannelMinimum (ChannelHandle *channelHandle*, f64 * *minimum*)

Get the minimum expected value for the signal connected to the channel.

Parameters:

channelHandle The handle to an existing channel

minimum The returned minimum value.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.143 UeiDaqAPI int UeiDaqGetAIChannelMuxPos (ChannelHandle *channelHandle*, tUeiAIChannelMuxPos * *muxPos*)

Get the mux position mode for devices with self-test capability

Parameters:

channelHandle The handle to an existing channel

muxPos The returned mux position mode

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.144 UeiDaqAPI int UeiDaqGetAIVExActualExcitationFrequency (ChannelHandle *channelHandle*, double * *pFex*)

Get the actual excitation frequency generated for the channel.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

pFex The excitation frequency

Returns:

The status code.

See also:

UeiDaqSetAIVExExcitationFrequency (p. 891),**UeiDaqGetAIVExExcitationFrequency** (p. 760)

3.2.3.145 UeiDaqAPI int UeiDaqGetAIVExActualShuntResistance (ChannelHandle *channelHandle*, double * *pActualResistance*)

Get the actual shunt resistance in Ohms. This resistance includes the programmed shunt resistance plus the resistance of other parts in the shunt circuitry.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

pActualResistance The actual shunt circuitry resistance.

Returns:

The status code.

See also:

UeiDaqSetAIVExShuntResistance (p. 893) **UeiDaqGetAIVExShuntResistance** (p. 763)

3.2.3.146 UeiDaqAPI int UeiDaqGetAIVExBridgeCompletionSetting (ChannelHandle *channelHandle*, double * *setting*)

Get the bridge completion setting used to program the bridge completion circuitry. Set it to 0.0 to automatically perform bridge completion next time the session is started. Make sure no strain is applied on the bridge.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

setting The current bridge completion setting.

Returns:

The status code.

See also:

UeiDaqSetAIVExBridgeCompletionSetting (p. 890)

3.2.3.147 UeiDaqAPI int UeiDaqGetAIVExBridgeType (ChannelHandle *channelHandle*, tUeiSensorBridgeType * *pBridgeType*)

Get the bridge type configured for the sensor connected to this channel.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

pBridgeType The bridge type

Returns:

The status code.

See also:

UeiDaqSetAIVExBridgeType (p. 891)

3.2.3.148 UeiDaqAPI int UeiDaqGetAIVExExcitationFrequency (ChannelHandle *channelHandle*, double * *pFex*)

Get the excitation frequency configured for the channel.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

pFex The excitation frequency

Returns:

The status code.

See also:

UeiDaqSetAIVExExcitationFrequency (p. 891)

3.2.3.149 UeiDaqAPI int UeiDaqGetAIVExExcitationVoltage (ChannelHandle *channelHandle*, double * *pVex*)

Get the excitation voltage configured for the channel.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

pVex The excitation voltage

Returns:

The status code.

See also:

UeiDaqSetAIVExExcitationVoltage (p. 891)

3.2.3.150 UeiDaqAPI int UeiDaqGetAIVExGainAdjustmentFactor (ChannelHandle *channelHandle*, double * *pGaf*)

Get the gain adjustment factor.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

pGaf The actual gain adjustment factor.

Returns:

The status code.

See also:

UeiDaqSetAIVExGainAdjustmentFactor (p. 892)

3.2.3.151 UeiDaqAPI int UeiDaqGetAIVExMeasuredExcitationVoltage (ChannelHandle *channelHandle*, double * *pVex*)

Get the actual excitation voltage measured by the channel.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

pVex The actual excitation voltage

Returns:

The status code.

See also:

UeiDaqGetAIVExExcitationVoltage (p. 761) **UeiDaqSetAIVExExcitationVoltage** (p. 891)

3.2.3.152 UeiDaqAPI int UeiDaqGetAIVExOffsetNullingSetting (ChannelHandle *channelHandle*, double * *setting*)

Get the offset nulling setting used to program the nulling circuitry. Set it to 0.0 to automatically perform offset nulling next time the session is started. Make sure no strain is applied on the bridge before nulling the offset.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

setting The current offset nulling setting.

Returns:

The status code.

See also:

UeiDaqSetAIVExOffsetNullingSetting (p. 892)

3.2.3.153 UeiDaqAPI int UeiDaqGetAIVExScalingWithExcitation (ChannelHandle *channelHandle*, int * *pScaleWithExcitation*)

Determines if acquired data will be divided by the excitation before being returned.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

pScaleWithExcitation The actual scaling mode

Returns:

The status code.

See also:

UeiDaqSetAIVExScalingWithExcitation (p. 892)

3.2.3.154 UeiDaqAPI int UeiDaqGetAIVExShuntLocation (ChannelHandle *channelHandle*, tUeiWheatstoneBridgeBranch * *pShuntedBranch*)

Get the branch of the wheatstone bridge on which the shunt resistor is connected.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

pShuntedBranch The actual shunted branch.

Returns:

The status code.

See also:

UeiDaqSetAIVExShuntLocation (p. 893)

3.2.3.155 UeiDaqAPI int UeiDaqGetAIVExShuntResistance (ChannelHandle *channelHandle*, double * *pResistance*)

Get the resistance programmed to the shunt resistor in Ohms. NOTE: This is not the actual resistance used to perform the calibration. The resistance of other components must be accounted for, use GetActualShuntResistance() to get the shunt resistance to use to calculate the gain adjustment factor.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

pResistance The actual resistance programmed on the shunt resistor.

Returns:

The status code.

See also:

UeiDaqSetAIVExShuntResistance (p. 893) **UeiDaqGetAIVExActualShuntResistance** (p. 759)

3.2.3.156 UeiDaqAPI int UeiDaqGetAIVExWiringScheme (ChannelHandle *channelHandle*, tUeiWiringScheme * *pWiring*)

Get the current wiring scheme used for this channel.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

pWiring The current wiring scheme.

Returns:

The status code.

See also:

UeiDaqSetAIVExWiringScheme (p. 894)

3.2.3.157 UeiDaqAPI int UeiDaqGetAllCircuitBreakerStatus (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numStatus*, uInt32 * *pBuffer*, Int32 * *numStatusRead*)

Read detailed status about each cicuit breaker. The meaning of each status bits depends on the type of the I/O device. Refer to the device's user manual for detailed description of each status bit

Parameters:

sessionHandle The handle to an existing session

port the port to get breaker status from

numStatus the number of status to read

pBuffer destination buffer for the channel(s) status

numStatusRead the actual number of status retrieved

Returns:

The status code

3.2.3.158 UeiDaqAPI int UeiDaqGetAOChannelDefaultValue (ChannelHandle *channelHandle*, f64 * *pDefaultVal*)

Get the default value.

Parameters:

channelHandle The handle to an existing channel

pDefaultVal the default value.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.159 UeiDaqAPI int UeiDaqGetAOChannelMaximum (ChannelHandle *channelHandle*, f64 * *maximum*)

Get the maximum value for the generated signal.

Parameters:

channelHandle The handle to an existing channel

maximum The returned maximum value.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.160 UeiDaqAPI int UeiDaqGetAOChannelMinimum (ChannelHandle *channelHandle*, f64 * *minimum*)

Get the minimum value for the generated signal.

Parameters:

channelHandle The handle to an existing channel

minimum The returned minimum value.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.161 UeiDaqAPI int UeiDaqGetAOProtectedADCChannel (ChannelHandle *channelHandle*, int *index*, tUeiAODiagChannel * *pAdcChannel*)

Each AO channel comes with up to five diagnostic channels. Select which diagnostic to monitor at specified index.

Parameters:

channelHandle The handle to an existing channel

index The 0 based index of the diagnostic channel

pAdcChannel The current diagnostic source.

Returns:

The status code

3.2.3.162 UeiDaqAPI int UeiDaqGetAOProtectedAutoRetry (ChannelHandle *channelHandle*, int * *pAutoRetry*)

The auto retry setting specifies wheter the device should attempt to close the circuit after it was opened because of an over or under limit condition. If it is set to true the device will try to close the circuit at a rate set by the autoRetryRate setting.

Parameters:

channelHandle The handle to an existing channel

pAutoRetry true if auto-retry is on, false otherwise.

Returns:

The status code

3.2.3.163 UeiDaqAPI int UeiDaqGetAOProtectedAutoRetryRate (ChannelHandle *channelHandle*, double * *pAutoRetryRate*)

Specifies how often the device will attemp to close a circuit that was opened because of an over or under limit condition.

Parameters:

channelHandle The handle to an existing channel

pAutoRetryRate The number of retries per second.

Returns:

The status code

3.2.3.164 UeiDaqAPI int UeiDaqGetAOProtectedCircuitBreakerHighLimit (ChannelHandle *channelHandle*, int *index*, double * *pHighLimit*)

Each circuit-breaker can monitor one diagnostic channel. Get the maximum current/volt/temperature allowed on the specified channel. The circuit will open if more than the maximum value is monitored.

Parameters:

channelHandle The handle to an existing channel

index The 0 based index of the circuit breaker

pHighLimit The current high limit.

Returns:

The status code

3.2.3.165 UeiDaqAPI int UeiDaqGetAOProtectedCircuitBreakerLowLimit (ChannelHandle *channelHandle*, int *index*, double * *pLowLimit*)

Each circuit-breaker can monitor one diagnostic channel. Get the minimum current/volt/temperature allowed on the specified channel. The circuit will open if less than the minimum value is monitored.

Parameters:

channelHandle The handle to an existing channel

index The 0 based index of the circuit breaker

pLowLimit The current low limit.

Returns:

The status code

3.2.3.166 UeiDaqAPI int UeiDaqGetAOProtectedDACMode (ChannelHandle *channelHandle*, tUeiAODACMode * *pMode*)

Each channel uses two DACs that can be enabled independantly.

Parameters:

channelHandle The handle to an existing channel

pMode The current DAC mode.

Returns:

The status code

3.2.3.167 UeiDaqAPI int UeiDaqGetAOProtectedMeasurementRate (ChannelHandle *channelHandle*, double * *pMeasurementRate*)

Get the rate at which the diagnostic channels are monitored. This rate determines how fast the device react when an under or over limit condition occurs.

Parameters:

channelHandle The handle to an existing channel

pMeasurementRate The current circuit breaker measurement rate.

Returns:

The status code

3.2.3.168 UeiDaqAPI int UeiDaqGetAOProtectedOverUnderCount (ChannelHandle *channelHandle*, uInt32 * *pOverUnderCount*)

Specifies number of consecutive over/under limit diagnostic readings that must occur in order to trip breaker.

Parameters:

channelHandle The handle to an existing channel

pOverUnderCount The maximum number of over/under readings.

Returns:

The status code

3.2.3.169 UeiDaqAPI int UeiDaqGetAOWaveformMainDACClockSource (ChannelHandle *channelHandle*, tUeiAOWaveformClockSource * *pSource*)

Get the source of the clock used by the main DAC.

Parameters:

channelHandle The handle to an existing channel

pSource the main DAC clock source.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.170 UeiDaqAPI int UeiDaqGetAOWaveformMainDACClockSync (ChannelHandle *channelHandle*, tUeiAOWaveformClockSync * *pSync*)

Get the output where the clock used by the main DAC is routed.

Parameters:

channelHandle The handle to an existing channel

pSync the main DAC clock synchronization.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.171 UeiDaqAPI int UeiDaqGetAOWaveformMainDACTriggerSource (ChannelHandle *channelHandle*, tUeiAOWaveformTriggerSource * *pSource*)

Get the source of the trigger used by the main DAC.

Parameters:

channelHandle The handle to an existing channel

pSource the main DAC trigger source.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.172 UeiDaqAPI int UeiDaqGetAOWaveformOffsetDACClockSource (ChannelHandle *channelHandle*, tUeiAOWaveformOffsetClockSource * *pSource*)

Get the source of the clock used by the offset DAC.

Parameters:

channelHandle The handle to an existing channel

pSource the offset DAC clock source.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.173 UeiDaqAPI int UeiDaqGetAOWaveformOffsetDACTriggerSource (ChannelHandle *channelHandle*, tUeiAOWaveformOffsetTriggerSource * *pSource*)

Get the source of the trigger used by the offset DAC.

Parameters:

channelHandle The handle to an existing channel

pSource the offset DAC trigger source.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.174 UeiDaqAPI int UeiDaqGetARINCInputFilterEntry (ChannelHandle *channelHandle*, int *index*, tUeiARINCFilterEntry ** *entry*)

Retrieve a filter entry from the port's frame filter table. Returns NULL when retrieving past the end of the table.

Parameters:

channelHandle The handle to an existing ARINC-429 input channel

index The index of the filter entry to retrieve

entry A pointer to the retrieved filter entry

Returns:

The status code.

See also:

UeiDaqAddARINCInputFilterEntry (p. 704), **UeiDaqClearARINCInputFilterEntries** (p. 707)

3.2.3.175 UeiDaqAPI int UeiDaqGetARINCInputParity (ChannelHandle *channelHandle*, tUeiARINCPortParity * *pParity*)

Get the parity used to detect transmission errors. The parity bit of each ARINC frame is set to 0 or 1 so that the number of bits set to 1 matches the specified parity.

Parameters:

channelHandle The handle to an existing ARINC-429 input channel

pParity The ARINC port parity

Returns:

The status code.

See also:

UeiDaqSetARINCInputParity (p. 900)

3.2.3.176 UeiDaqAPI int UeiDaqGetARINCInputSDIFilterMask (ChannelHandle *channelHandle*, uInt32 * *pSDIFilterMask*)

Get the SDI filter mask, only bits 0 and 1 are meaningful. ARINC frames whose SDI bits don't match the SDI mask are rejected.

Parameters:

channelHandle The handle to an existing ARINC-429 input channel

pSDIFilterMask The current SDI filter mask

Returns:

The status code.

See also:

UeiDaqSetARINCInputSDIFilterMask (p. 900)

3.2.3.177 UeiDaqAPI int UeiDaqGetARINCInputSpeed (ChannelHandle *channelHandle*, tUeiARINCPortSpeed * *pBitsPerSecond*)

Get the number of bits transmitted per second.

Parameters:

channelHandle The handle to an existing ARINC-429 input channel

pBitsPerSecond The ARINC port speed

Returns:

The status code.

See also:

UeiDaqSetARINCInputSpeed (p. 900)

3.2.3.178 UeiDaqAPI int UeiDaqGetARINCOutputFIFORate (ChannelHandle *channelHandle*, double * *rate*)

Gets TX FIFO rate.

Parameters:

channelHandle The handle to an existing ARINC-429 output channel

rate The new TX FIFO rate.

Returns:

The status code.

See also:

UeiDaqSetARINCOutputFIFORate (p. 901)

3.2.3.179 UeiDaqAPI int UeiDaqGetARINCOutputMinorFrameEntry (ChannelHandle *channelHandle*, int *index*, tUeiARINCMinorFrameEntry ** *entry*)

Retrieve a minor frame entry from the port's minor frame table. Returns NULL when retrieving past the end of the table.

Parameters:

channelHandle The handle to an existing ARINC-429 output channel

index The index of the minor frame entry to retrieve

entry A pointer to the retrieved minor frame entry

Returns:

The status code.

See also:

UeiDaqAddARINCOutputMinorFrameEntry (p. 705), **UeiDaqClearARINCOutputMinorFrameEntries** (p. 707)

3.2.3.180 UeiDaqAPI int UeiDaqGetARINCOutputParity (ChannelHandle *channelHandle*, tUeiARINCPortParity * *pParity*)

Get the parity used to detect transmission errors. The parity bit of each ARINC frame is set to 0 or 1 so that the number of bits set to 1 matches the specified parity.

Parameters:

channelHandle The handle to an existing ARINC-429 output channel

pParity The ARINC port parity

Returns:

The status code.

See also:

UeiDaqSetARINCOutputParity (p. 901)

3.2.3.181 UeiDaqAPI int UeiDaqGetARINCOutputSchedulerEntry (ChannelHandle *channelHandle*, int *index*, tUeiARINCSchedulerEntry ** *entry*)

Retrieve a scheduler entry from the port's scheduler table. Returns NULL when retrieving past the end of the table.

Parameters:

channelHandle The handle to an existing ARINC-429 output channel

index The index of the scheduler entry to retrieve

entry A pointer to the retrieved scheduler entry

Returns:

The status code.

See also:

UeiDaqAddARINCOutputSchedulerEntry (p. 705), **UeiDaqClearARINCOutputSchedulerEntries** (p. 708)

3.2.3.182 UeiDaqAPI int UeiDaqGetARINCOutputSchedulerRate (ChannelHandle *channelHandle*, double * *rate*)

Gets scheduler rate.

Parameters:

channelHandle The handle to an existing ARINC-429 output channel

rate The current scheduler rate.

Returns:

The status code.

See also:

UeiDaqSetARINCOutputSchedulerRate (p. 901)

3.2.3.183 UeiDaqAPI int UeiDaqGetARINCOutputSchedulerType (ChannelHandle *channelHandle*, tUeiARINCSchedulerType * *type*)

Gets scheduler type.

Parameters:

channelHandle The handle to an existing ARINC-429 output channel

type The current scheduler type.

Returns:

The status code.

See also:

UeiDaqSetARINCOutputSchedulerType (p. 902)

3.2.3.184 UeiDaqAPI int UeiDaqGetARINCOutputSpeed (ChannelHandle *channelHandle*, tUeiARINCPortSpeed * *pBitsPerSecond*)

Get the number of bits transmitted per second.

Parameters:

channelHandle The handle to an existing ARINC-429 output channel

pBitsPerSecond The ARINC port speed

Returns:

The status code.

See also:

UeiDaqSetARINCOutputSpeed (p. 902)

3.2.3.185 UeiDaqAPI Int32 UeiDaqGetAvailableInputMessages (SessionHandle *sessionHandle*, Int32 *port*, Int32 * *pAvailableInputMessages*)

Get the number of messages ready to be read from the specified port

Parameters:

sessionHandle The handle to an existing session

port the port to query

pAvailableInputMessages the number of available messages.

Returns:

The status code

3.2.3.186 UeiDaqAPI Int32 UeiDaqGetAvailableOutputSlots (SessionHandle *sessionHandle*, Int32 *port*, Int32 * *pAvailableOutputSlots*)

Get the number of message slots ready to accept new output messages on the specified port

Parameters:

sessionHandle The handle to an existing session

port the port to query

pAvailableOutputSlots the number of available messages.

Returns:

The status code

3.2.3.187 UeiDaqAPI int UeiDaqGetCANPortAcceptanceCode (ChannelHandle *channelHandle*, uInt32 * *pCode*)

The acceptance code is used to filter incoming frames. The arbitration ID bits selected by the mask are compared to the code and the frame is rejected if there is any difference. If (mask XOR id) AND code == 0 the frame is accepted

Parameters:

channelHandle The handle to an existing CAN port

pCode The current acceptance code.

Returns:

The status code.

See also:

UeiDaqSetCANPortAcceptanceCode (p. 903)

3.2.3.188 UeiDaqAPI int UeiDaqGetCANPortAcceptanceMask (ChannelHandle *channelHandle*, uInt32 * *pMask*)

The acceptance mask is used to filter incoming frames. The mask selects which bits within arbitration ID will be used for filtering.

Parameters:

channelHandle The handle to an existing CAN port

pMask The current acceptance mask.

Returns:

The status code.

See also:

UeiDaqSetCANPortAcceptanceMask (p. 903)

3.2.3.189 UeiDaqAPI int UeiDaqGetCANPortArbitrationLostCaptureRegister (ChannelHandle *channelHandle*, uInt32 * *pArbLostCaptureReg*)

Get the current values of the Arbitration Lost Capture register from the NXP SJA1000 CAN controller chip.

The Arbitration Lost Capture register provides information on a loss of arbitration during transmits. Loss of arbitration is not considered an error. When communication starts on the interface, the first arbitration loss is captured into the Arbitration Lost Capture register, and retained until you get this value. Then, the Arbitration Lost Capture register is again enabled to capture information for the next arbitration loss.

Parameters:

channelHandle The handle to an existing CAN port

pArbLostCaptureReg The arbitration lost capture register.

Returns:

The status code.

3.2.3.190 UeiDaqAPI int UeiDaqGetCANPortBitTimingRegisters (ChannelHandle *channelHandle*, uInt16 * *pBtrValues*)

Get the current values of the bit timing registers from the NXP SJA1000 CAN controller chip. BTR0 and BTR1 set the baud rate for the CAN port. For information on CAN bit timing registers, refer to the datasheet of the NXP SJA1000 CAN controller, available for download on www.nxp.com.

Parameters:

channelHandle The handle to an existing CAN port

pBtrValues The bit timing regiters, BTR0 is LSB and BTR1 is MSB.

Returns:

The status code.

3.2.3.191 UeiDaqAPI int UeiDaqGetCANPortErrorCodeCaptureRegister (ChannelHandle *channelHandle*, uInt32 * *pErrorCodeCaptureReg*)

Get the current values of the Error Code Capture register from the NXP SJA1000 CAN controller chip. The Error Code Capture register provides information on bus errors that occur according to the CAN standard. A bus error increments either the Transmit Error Counter or the Receive Error Counter. When communication starts on the interface, the first bus error is captured into the Error Code Capture register, and retained until you get this value. Then the Error Code Capture register is again enabled to capture information for the next bus error.

Parameters:

channelHandle The handle to an existing CAN port

pErrorCodeCaptureReg The error code capture register.

Returns:

The status code.

3.2.3.192 UeiDaqAPI int UeiDaqGetCANPortFilterEntry (ChannelHandle *channelHandle*, int *index*, tUeiCANFilterEntry ** *entry*)

Retrieve a filter entry from the port's frame filter table. Returns NULL when retrieving past the end of the table.

Parameters:

channelHandle The handle to an existing CAN port

index The index of the filter entry to retrieve

entry A pointer to the retrieved filter entry

Returns:

The status code.

See also:

UeiDaqAddCANPortFilterEntry (p. 706), **UeiDaqClearCANPortFilterEntries** (p. 708)

3.2.3.193 UeiDaqAPI int UeiDaqGetCANPortFrameFormat (ChannelHandle *channelHandle*, tUeiCANFrameFormat * *pFrameFormat*)

Get the format of frames transmitted by this port

Parameters:

channelHandle The handle to an existing CAN port

pFrameFormat The frame format.

Returns:

The status code.

See also:

UeiDaqSetCANPortFrameFormat (p. 904)

3.2.3.194 UeiDaqAPI int UeiDaqGetCANPortMode (ChannelHandle *channelHandle*, tUeiCANPortMode * *pMode*)

Get the CAN port operation mode. Possible values are normal and passive

Parameters:

channelHandle The handle to an existing CAN port

pMode The operation mode.

Returns:

The status code.

See also:

UeiDaqSetCANPortMode (p. 904)

3.2.3.195 UeiDaqAPI int UeiDaqGetCANPortReceiveErrorCounter (ChannelHandle *channelHandle*, uInt32 * *pRxErrorCounter*)

Get the number of receive errors detected on the bus since the session started.

Parameters:

channelHandle The handle to an existing CAN port

pRxErrorCounter The receive error counter.

Returns:

The status code.

See also:

UeiDaqGetCANPortTransmitErrorCounter (p. 777)

3.2.3.196 UeiDaqAPI int UeiDaqGetCANPortSpeed (ChannelHandle *channelHandle*, tUeiCANPortSpeed * *pBitsPerSecond*)

Get the number of data bits transmitted per second.

Parameters:

channelHandle The handle to an existing CAN port

pBitsPerSecond the CAN port speed.

Returns:

The status code.

See also:

UeiDaqSetCANPortSpeed (p. 904)

3.2.3.197 UeiDaqAPI int UeiDaqGetCANPortTransmitErrorCounter (ChannelHandle *channelHandle*, uInt32 * *pTxErrorCounter*)

Get the number of transmit errors detected on the bus since the session started.

Parameters:

channelHandle The handle to an existing CAN port

pTxErrorCounter The transmit error counter.

Returns:

The status code.

See also:

UeiDaqGetCANPortReceiveErrorCounter (p. 776)

3.2.3.198 UeiDaqAPI int UeiDaqGetChannelAliasName (ChannelHandle *channelHandle*, char * *aliasName*, int * *aliasNameLength*)

Get the alias name of the channel

Parameters:

channelHandle The handle to an existing channel

aliasName A string to contain the returned alias name.

aliasNameLength The length of the string, on function return contains the number of characters copied. If aliasName is NULL, aliasNameLength contains the required length for aliasName

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.199 UeiDaqAPI int UeiDaqGetChannelHandle (SessionHandle *sessionHandle*, int *index*, ChannelHandle * *channel*)

Get the handle on the channel object specified by its index in the channel list.

Parameters:

sessionHandle The handle to an existing session

index index of the channel object in the channel list

channel A handle to the channel object.

Returns:

The status code

3.2.3.200 UeiDaqAPI int UeiDaqGetChannelHandlelById (SessionHandle *sessionHandle*, int *id*, ChannelHandle * *channel*)

Get thehandle on a channel object specified by its physical channel id.

Parameters:

sessionHandle The handle to an existing session

id physical id of the channel object

channel A handle to the channel object.

Returns:

The status code

3.2.3.201 UeiDaqAPI int UeiDaqGetChannelIndex (ChannelHandle *channelHandle*, int * *index*)

Get the index of the channel in the session's channel list

Parameters:

channelHandle The handle to an existing channel

index The returned channel index.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.202 UeiDaqAPI int UeiDaqGetChannelResourceName (ChannelHandle *channelHandle*, char * *resourceName*, int * *resourceNameLength*)

Get the channel URL.

Parameters:

channelHandle The handle to an existing channel

resourceName A string to contain the returned resource name.

resourceNameLength The length of the string, on function return contains the number of characters copied. If resourceName is NULL, resourceNameLength contains the required length for resourceName

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.203 UeiDaqAPI int UeiDaqGetCIChannelCounterGate (ChannelHandle *channelHandle*, tUeiCounterGate * *gate*)

Get the signal that specify whether the counter/timer is on or off

Parameters:

channelHandle The handle to an existing channel

gate The returned gate.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.204 UeiDaqAPI int UeiDaqGetCIChannelCounterMode (ChannelHandle *channelHandle*, tUeiCounterMode * *mode*)

Get the mode that specify whether the session counts events, measures a pulse width or measures a period on the signal configured as the source

Parameters:

channelHandle The handle to an existing channel

mode The returned mode.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.205 UeiDaqAPI int UeiDaqGetCIChannelCounterSource (ChannelHandle *channelHandle*, tUeiCounterSource * *source*)

Get the source of the signal counted or used as a timebase

Parameters:

channelHandle The handle to an existing channel

source The returned source.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.206 UeiDaqAPI int UeiDaqGetCIChannelInverted (ChannelHandle *channelHandle*, Int32 * *inverted*)

Checks whether the signal at the source is inverted before performing the counting operation

Parameters:

channelHandle The handle to an existing channel

inverted The returned inverted setting.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.207 UeiDaqAPI int UeiDaqGetCIGateMode (ChannelHandle *channelHandle*, tUeiCounterGateMode * *gateMode*)

Get the gate mode which determines whether the counter/timer will run continuously once the gate is asserted

Parameters:

channelHandle The handle to an existing channel

gateMode the current gate mode.

Returns:

The status code

See also:

UeiDaqSetCIGateMode (p. 907)

3.2.3.208 UeiDaqAPI int UeiDaqGetCIMinimumGatePulseWidth (ChannelHandle *channelHandle*, double * *minWidth*)

Get the minimum pulse width in ms the counter recognizes at its gate. Any pulse whose width is smaller will be ignored.

Parameters:

channelHandle The handle to an existing channel

minWidth the current minimum pulse width

Returns:

The status code

See also:

UeiDaqSetCIMinimumGatePulseWidth (p. 908)

3.2.3.209 UeiDaqAPI int UeiDaqGetCIMinimumSourcePulseWidth (ChannelHandle *channelHandle*, double * *minWidth*)

Get the minimum pulse width in ms the counter recognizes at its source. Any pulse whose width is smaller will be ignored.

Parameters:

channelHandle The handle to an existing channel

minWidth the current minimum pulse width

See also:

UeiDaqSetCIMinimumSourcePulseWidth (p. 908)

3.2.3.210 UeiDaqAPI int UeiDaqGetCIPeriodCount (ChannelHandle *channelHandle*, Int32 * *periodCount*)

Get the number of periods used to average a period measurement

Parameters:

channelHandle The handle to an existing channel

periodCount the current period count.

Returns:

The status code

See also:

UeiDaqSetCIPeriodCount (p. 908)

3.2.3.211 UeiDaqAPI int UeiDaqGetCircuitBreakerStatus (SessionHandle *sessionHandle*, Int32 *port*, uInt32 * *currentStatus*, uInt32 * *stickyStatus*)

Get circuit breaker status masks. Each bit in the current status mask correspond to a circuit breaker. 1 if CB is currently tripped, 0 otherwise Each bit in the sticky status mask correspond to a circuit breaker. 1 if CB was tripped at least once since last time status was read, 0 otherwise

Parameters:

sessionHandle The handle to an existing session

port the port to get breaker status from

currentStatus the current circuit breaker status bits (1 if tripped, 0 otherwise)

stickyStatus the sticky circuit breaker status bits (1 if tripped, 0 otherwise)

Returns:

The status code

3.2.3.212 UeiDaqAPI int UeiDaqGetCITimebaseDivider (ChannelHandle *channelHandle*, Int32 * *divider*)

Get the divider used to scale the timebase speed.

Parameters:

channelHandle The handle to an existing channel

divider the current divider.

Returns:

The status code

See also:

UeiDaqSetCITimebaseDivider (p. 909)

3.2.3.213 UeiDaqAPI int UeiDaqGetCOChannelCounterGate (ChannelHandle *channelHandle*, tUeiCounterGate * *gate*)

Get the signal that specify whether the counter/timer is on or off

Parameters:

channelHandle The handle to an existing channel

gate The returned gate.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.214 UeiDaqAPI int UeiDaqGetCOChannelCounterMode (ChannelHandle *channelHandle*, tUeiCounterMode * *mode*)

Get the mode that specify whether the session generates a single pulse or a pulse train on the counter output

Parameters:

channelHandle The handle to an existing channel

mode The returned mode.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.215 UeiDaqAPI int UeiDaqGetCOChannelCounterSource (ChannelHandle *channelHandle*, tUeiCounterSource * *source*)

Get the source of the signal used as a timebase

Parameters:

channelHandle The handle to an existing channel

source The returned source.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.216 UeiDaqAPI int UeiDaqGetCOChannelInverted (ChannelHandle *channelHandle*, Int32 * *inverted*)

Checks whether the signal at the source is inverted before performing the timing operation

Parameters:

channelHandle The handle to an existing channel

inverted The returned inverted setting.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.217 UeiDaqAPI int UeiDaqGetCOChannelTick1 (ChannelHandle *channelHandle*, Int32 * *tick1*)

Specifies how long the output stays low.

Parameters:

channelHandle The handle to an existing channel

tick1 The returned number of ticks of the signal connected at the source.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.218 UeiDaqAPI int UeiDaqGetCOChannelTick2 (ChannelHandle *channelHandle*, Int32 * *tick2*)

Specifies how long the output stays high.

Parameters:

channelHandle The handle to an existing channel

tick2 The returned number of ticks of the signal connected at the source.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.219 UeiDaqAPI int UeiDaqGetCOGateMode (ChannelHandle *channelHandle*, tUeiCounterGateMode * *gateMode*)

Get the gate mode which determines whether the counter/timer will run continuously once the gate is asserted

Parameters:

channelHandle The handle to an existing channel

gateMode the current gate mode.

Returns:

The status code

See also:

UeiDaqSetCOGateMode (p. 911)

3.2.3.220 UeiDaqAPI int UeiDaqGetCOMinimumGatePulseWidth (ChannelHandle *channelHandle*, double * *minWidth*)

Get the minimum pulse width in ms the counter recognizes at its gate. Any pulse whose width is smaller will be ignored.

Parameters:

channelHandle The handle to an existing channel

minWidth the current minimum pulse width

Returns:

The status code

See also:

UeiDaqSetCOMinimumGatePulseWidth (p. 911)

3.2.3.221 UeiDaqAPI int UeiDaqGetCOMinimumSourcePulseWidth (ChannelHandle *channelHandle*, double * *minWidth*)

Get the minimum pulse width in ms the counter recognizes at its source. Any pulse whose width is smaller will be ignored.

Parameters:

channelHandle The handle to an existing channel

minWidth the current minimum pulse width

See also:

UeiDaqSetCOMinimumSourcePulseWidth (p. 912)

3.2.3.222 UeiDaqAPI int UeiDaqGetCONumberOfPulses (ChannelHandle *channelHandle*, Int32 * *numberOfPulses*)

Get the number of pulses to generate (-1 for continuous)

Parameters:

channelHandle The handle to an existing channel

numberOfPulses the current number of pulses.

Returns:

The status code

See also:

SetNumberOfPulses

3.2.3.223 UeiDaqAPI int UeiDaqGetCOTimebaseDivider (ChannelHandle *channelHandle*, Int32 * *divider*)

Get the divider used to scale the timebase speed.

Parameters:

channelHandle The handle to an existing channel

divider the current divider.

Returns:

The status code

See also:

UeiDaqSetCOTimebaseDivider (p. 912)

3.2.3.224 UeiDaqAPI int UeiDaqGetDataStreamAvailableScans (DataStreamHandle *dataStreamHandle*, Int32 * *availableScans*)

Get the number of scans ready to be read or written

Parameters:

dataStreamHandle The handle to an existing data stream

availableScans The returned number of available scans.

Returns:

The status code

See also:

UeiDaqGetDataStreamHandle (p. 787).

3.2.3.225 UeiDaqAPI int UeiDaqGetDataStreamBurst (DataStreamHandle *dataStreamHandle*, Int32 * *pBurst*)

Get the parameter that determines whether the input device should acquire all the requested data in its internal memory before transferring it to the host PC.

Parameters:

dataStreamHandle The handle to an existing data stream

pBurst the current burst setting value.

See also:

UeiDaqSetDataStreamBurst (p. 913)

3.2.3.226 UeiDaqAPI int UeiDaqGetDataStreamCurrentScan (DataStreamHandle *dataStreamHandle*, Int32 * *currentScan*)

Get the latest acquired scan position in the acquisition buffer.

Parameters:

dataStreamHandle The handle to an existing data stream

currentScan The returned current scan position.

Returns:

The status code

See also:

UeiDaqGetDataStreamHandle (p. 787).

3.2.3.227 UeiDaqAPI int UeiDaqGetDataStreamHandle (SessionHandle *sessionHandle*, DataStreamHandle * *stream*)

Get a handle to the stream object associated with the session.

Parameters:

sessionHandle The handle to an existing session

stream A handle to the DataStream object that is used to access data.

Returns:

The status code

3.2.3.228 UeiDaqAPI int UeiDaqGetDataStreamNumberOfFrames (DataStreamHandle *dataStreamHandle*, Int32 * *numFrames*)

Get the number of frames. Each frame is accessed independently by the device.

Parameters:

dataStreamHandle The handle to an existing data stream

numFrames The returned number of frames.

Returns:

The status code

See also:

UeiDaqGetDataStreamHandle (p. 787).

3.2.3.229 UeiDaqAPI int UeiDaqGetDataStreamNumberOfScans (DataStreamHandle *dataStreamHandle*, Int32 * *numScans*)

Get the number of scans to read at a time

Parameters:

dataStreamHandle The handle to an existing data stream

numScans The returned number of scans.

Returns:

The status code

See also:

UeiDaqGetDataStreamHandle (p. 787)

3.2.3.230 UeiDaqAPI int UeiDaqGetDataStreamOverUnderRun (DataStreamHandle *dataStreamHandle*, Int32 * *overunderrun*)

For an input session, Determines whether the device overwrite acquired data if it was not read fast enough. For an output session, Determines whether the device regenerate previously generated data if it didn't receive new data fast enough.

Parameters:

dataStreamHandle The handle to an existing data stream

overunderrun The returned overrun/underrun setting.

Returns:

The status code

See also:

UeiDaqGetDataStreamHandle (p. 787).

3.2.3.231 UeiDaqAPI int UeiDaqGetDataStreamRegenerate (DataStreamHandle *dataStreamHandle*, Int32 * *regenerate*)

Get the parameter that determines whether the output device continuously regenerate the first buffer it received.

Parameters:

dataStreamHandle The handle to an existing data stream

regenerate The regenerate setting.

Returns:

The status code

See also:

UeiDaqGetDataStreamHandle (p. 787).

3.2.3.232 UeiDaqAPI int UeiDaqGetDataStreamRelativeTo (DataStreamHandle *dataStreamHandle*, tUeiDataStreamRelativeTo * *relativeTo*)

Get the position used as a reference in the framework's internal buffer

Parameters:

dataStreamHandle The handle to an existing data stream

relativeTo The returned reference.

Returns:

The status code

See also:

UeiDaqGetDataStreamHandle (p. 787).

3.2.3.233 UeiDaqAPI int UeiDaqGetDataStreamSampleSize (DataStreamHandle *dataStreamHandle*, Int32 * *sampleSize*)

Get the size of a raw sample in bytes.

Parameters:

dataStreamHandle The handle to an existing data stream

sampleSize The returned sample size.

Returns:

The status code

See also:

UeiDaqGetDataStreamHandle (p. 787).

3.2.3.234 UeiDaqAPI int UeiDaqGetDataStreamScanSize (DataStreamHandle *dataStreamHandle*, Int32 * *scanSize*)

Get the size of a scan, usually equal to the number of channels in the session's channel list.

Parameters:

dataStreamHandle The handle to an existing data stream

scanSize The returned number of scans.

Returns:

The status code

See also:

UeiDaqGetDataStreamHandle (p. 787).

3.2.3.235 UeiDaqAPI int UeiDaqGetDataStreamTotalScans (DataStreamHandle *dataStreamHandle*, Int32 * *totalScans*)

Get the total number of scans read or written since the session started.

Parameters:

dataStreamHandle The handle to an existing data stream

totalScans The returned number of total scans.

Returns:

The status code

See also:

UeiDaqGetDataStreamHandle (p. 787).

3.2.3.236 UeiDaqAPI int UeiDaqGetDeviceAIDataSize (DeviceHandle *deviceHandle*, Int32 * *dataSize*)

Get the number of bytes needed to store each Analog input sample

Parameters:

deviceHandle The handle to an existing device

dataSize The number of bytes per sample

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.237 UeiDaqAPI int UeiDaqGetDeviceAIResolution (DeviceHandle *deviceHandle*, Int32 * *resolution*)

Get the resolution of the Analog Input subsystem

Parameters:

deviceHandle The handle to an existing device

resolution The returned Analog Input resolution

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.238 UeiDaqAPI int UeiDaqGetDeviceAISimultaneous (DeviceHandle *deviceHandle*, Int32 * *simultaneous*)

Check if the device can do simultaneous analog input sampling

Parameters:

deviceHandle The handle to an existing device

simultaneous Set to 1 if the device supports simultaneous analog input

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.239 UeiDaqAPI int UeiDaqGetDeviceAODataSize (DeviceHandle *deviceHandle*, Int32 * *dataSize*)

Get the number of bytes needed to store each Analog output sample

Parameters:

deviceHandle The handle to an existing device

dataSize The number of bytes per sample

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.240 UeiDaqAPI int UeiDaqGetDeviceAOResolution (DeviceHandle *deviceHandle*, Int32 * *resolution*)

Get the resolution of the Analog Output subsystem

Parameters:

deviceHandle The handle to an existing device

resolution The returned Analog Output resolution

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.241 UeiDaqAPI int UeiDaqGetDeviceAOSimultaneous (DeviceHandle *deviceHandle*, Int32 * *simultaneous*)

Check if the device can do simultaneous analog output update

Parameters:

deviceHandle The handle to an existing device

simultaneous Set to 1 if the device supports simultaneous analog output

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.242 UeiDaqAPI int UeiDaqGetDeviceBidirectionalDigitalPorts (DeviceHandle *deviceHandle*, Int32 * *isBidirectional*)

Check if the device digital ports are bidirectional

Parameters:

deviceHandle The handle to an existing device

isBidirectional set to 1 if the device's digital ports are bidirectional

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.243 UeiDaqAPI int UeiDaqGetDeviceCanRegenerate (DeviceHandle *deviceHandle*, Int32 * *canRegenerate*)

Check if the device supports output regeneration

Parameters:

deviceHandle The handle to an existing device

canRegenerate set to 1 if the device can regenerate on its analog outputs

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.244 UeiDaqAPI int UeiDaqGetDeviceCIDataSize (DeviceHandle *deviceHandle*, Int32 * *dataSize*)

Get the number of bytes needed to store each counter input sample

Parameters:

deviceHandle The handle to an existing device

dataSize The number of bytes per sample

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.245 UeiDaqAPI int UeiDaqGetDeviceCIResolution (DeviceHandle *deviceHandle*, Int32 * *resolution*)

Get the resolution of the Counter/Timer Input subsystem

Parameters:

deviceHandle The handle to an existing device

resolution The returned Counter Input resolution

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.246 UeiDaqAPI int UeiDaqGetDeviceCODataSize (DeviceHandle *deviceHandle*, Int32 * *dataSize*)

Get the number of bytes needed to store each counter output sample

Parameters:

deviceHandle The handle to an existing device

dataSize The number of bytes per sample

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.247 UeiDaqAPI int UeiDaqGetDeviceCOResolution (DeviceHandle *deviceHandle*, Int32 * *resolution*)

Get the resolution of the Counter/Timer Output subsystem

Parameters:

deviceHandle The handle to an existing device

resolution The returned Timer Output resolution

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.248 UeiDaqAPI int UeiDaqGetDeviceDIDataSize (DeviceHandle *deviceHandle*, Int32 * *dataSize*)

Get the number of bytes needed to store each Digital input sample

Parameters:

deviceHandle The handle to an existing device

dataSize The number of bytes per sample

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.249 UeiDaqAPI int UeiDaqGetDeviceDIResolution (DeviceHandle *deviceHandle*, Int32 * *resolution*)

Get the resolution of the Digital Input subsystem

Parameters:

deviceHandle The handle to an existing device

resolution The returned Digital Input resolution

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.250 UeiDaqAPI int UeiDaqGetDeviceDODataSize (DeviceHandle *deviceHandle*, Int32 * *dataSize*)

Get the number of bytes needed to store each Digital output sample

Parameters:

deviceHandle The handle to an existing device

dataSize The number of bytes per sample

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.251 UeiDaqAPI int UeiDaqGetDeviceDOResolution (DeviceHandle *deviceHandle*, Int32 * *resolution*)

Get the resolution of the Digital Output subsystem

Parameters:

deviceHandle The handle to an existing device

resolution The returned Digital Output resolution

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.252 UeiDaqAPI int UeiDaqGetDeviceHandle (SessionHandle *sessionHandle*, DeviceHandle * *device*)

Get a handle to the device object associated with the session.

Parameters:

sessionHandle The handle to an existing session

device A handle to the Device object that can be later used to retrieve information about the device associated to the session.

Returns:

The status code

3.2.3.253 UeiDaqAPI int UeiDaqGetDeviceHardwareInformation (DeviceHandle *deviceHandle*, char * *info*, int * *infoLength*)

Get a string containing low-level information about the device

Parameters:

deviceHandle The handle to an existing device

info the hardware information string

infoLength The length of the info string, on function return contains the number of characters copied. If info is NULL, infoLength contains the required length for info

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.254 UeiDaqAPI int UeiDaqGetDeviceHighRanges (DeviceHandle *deviceHandle*, tUeiSessionType *type*, int *subsystem*, Int32 * *numValues*, double * *highValues*)

Get the high values for specifed AI subsystem

Parameters:

deviceHandle The handle to an existing device

type The session type

subsystem The index of the subsystem (for devices with multiple sub-systems)

numValues The values array capacity as input, actual number of values copied as output

highValues Array containing the high range values

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.255 UeiDaqAPI int UeiDaqGetDeviceIndex (DeviceHandle *deviceHandle*, Int32 * *index*)

Get the index of this device in the device class

Parameters:

deviceHandle The handle to an existing device

index The returned device index

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.256 UeiDaqAPI int UeiDaqGetDeviceInputFIFOSize (DeviceHandle *deviceHandle*, Int32 * *inputFIFOSize*)

Get the size of the input FIFO in number of samples

Parameters:

deviceHandle The handle to an existing device
inputFIFOSize the number of samples that can be stored in the input FIFO

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.257 UeiDaqAPI int UeiDaqGetDeviceLowRanges (DeviceHandle *deviceHandle*, tUeiSessionType *type*, int *subsystem*, Int32 * *numValues*, double * *lowValues*)

Get the low values for specifed AI subsystem

Parameters:

deviceHandle The handle to an existing device
type The session type
subsystem The index of the subsystem (for devices with multiple sub-systems)
numValues The values array capacity as input, actual number of values copied as output
lowValues Array containing the low range values

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.258 UeiDaqAPI int UeiDaqGetDeviceMaxAIRate (DeviceHandle *deviceHandle*, Int32 * *maxRate*)

Get the maximum rate of the Analog Input subsystem

Parameters:

deviceHandle The handle to an existing device
maxRate The returned maximum Analog Input rate

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.259 UeiDaqAPI int UeiDaqGetDeviceMaxAORate (DeviceHandle *deviceHandle*, Int32 * *maxRate*)

Get the maximum rate of the Analog Output subsystem

Parameters:

deviceHandle The handle to an existing device

maxRate The returned maximum Analog Output rate

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.260 UeiDaqAPI int UeiDaqGetDeviceMaxCIRate (DeviceHandle *deviceHandle*, Int32 * *maxRate*)

Get the maximum rate of the Counter Input subsystem

Parameters:

deviceHandle The handle to an existing device

maxRate The returned maximum Counter Input rate

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.261 UeiDaqAPI int UeiDaqGetDeviceMaxCORate (DeviceHandle *deviceHandle*, Int32 * *maxRate*)

Get the maximum rate of the Counter Output subsystem

Parameters:

deviceHandle The handle to an existing device

maxRate The returned maximum Counter Output rate

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.262 UeiDaqAPI int UeiDaqGetDeviceMaxDIRate (DeviceHandle *deviceHandle*, Int32 * *maxRate*)

Get the maximum rate of the Digital Input subsystem

Parameters:

deviceHandle The handle to an existing device

maxRate The returned maximum Digital Input rate

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.263 UeiDaqAPI int UeiDaqGetDeviceMaxDORate (DeviceHandle *deviceHandle*, Int32 * *maxRate*)

Get the maximum rate of the Digital Output subsystem

Parameters:

deviceHandle The handle to an existing device

maxRate The returned maximum Digital Output rate

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.264 UeiDaqAPI int UeiDaqGetDeviceName (DeviceHandle *deviceHandle*, char * *deviceName*, int * *deviceNameLength*)

Get the name from the device database

Parameters:

deviceHandle The handle to an existing device

deviceName A string to contain the returned device name.

deviceNameLength The length of the string, on function return contains the number of characters copied. If deviceName is NULL, deviceNameLength contains the required length for deviceName

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.265 UeiDaqAPI int UeiDaqGetDeviceNumberOfAIDifferentialChannels (DeviceHandle *deviceHandle*, Int32 * *numChannels*)

Get the number of Analog Input differential channels

Parameters:

deviceHandle The handle to an existing device

numChannels The returned number of Analog Input differential channels

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.266 UeiDaqAPI int UeiDaqGetDeviceNumberOfAISingleEndedChannels (DeviceHandle *deviceHandle*, Int32 * *numChannels*)

Get the number of Analog Input single-ended channels

Parameters:

deviceHandle The handle to an existing device

numChannels The returned number of Analog Input single-ended channels

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.267 UeiDaqAPI int UeiDaqGetDeviceNumberOfAOChannels (DeviceHandle *deviceHandle*, Int32 * *numChannels*)

Get the number of Analog Output channels

Parameters:

deviceHandle The handle to an existing device

numChannels The returned number of Analog Outout channels

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.268 UeiDaqAPI int UeiDaqGetDeviceNumberOfARINCInputPorts (DeviceHandle *deviceHandle*, Int32 * *numARINCInputPorts*)

Get the number of ARINC-429 input ports

Parameters:

deviceHandle The handle to an existing device

numARINCInputPorts The returned number of ARINC-429 input ports

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.269 UeiDaqAPI int UeiDaqGetDeviceNumberOfARINCOutputPorts (DeviceHandle *deviceHandle*, Int32 * *numARINCOutputPorts*)

Get the number of ARINC-429 output ports

Parameters:

deviceHandle The handle to an existing device

numARINCOutputPorts The returned number of ARINC-429 output ports

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.270 UeiDaqAPI int UeiDaqGetDeviceNumberOfCANPorts (DeviceHandle *deviceHandle*, Int32 * *numCANPorts*)

Get the number of CAN ports

Parameters:

deviceHandle The handle to an existing device

numCANPorts The returned number of CAN ports

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.271 UeiDaqAPI int UeiDaqGetDeviceNumberOfCIChannels (DeviceHandle *deviceHandle*, Int32 * *numChannels*)

Get the number of Counter Input channels

Parameters:

deviceHandle The handle to an existing device

numChannels The returned number of Counter Input channels

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.272 UeiDaqAPI int UeiDaqGetDeviceNumberOfCOChannels (DeviceHandle *deviceHandle*, Int32 * *numChannels*)

Get the number of Counter Output channels

Parameters:

deviceHandle The handle to an existing device

numChannels The returned number of Counter Output channels

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.273 UeiDaqAPI int UeiDaqGetDeviceNumberOfDIChannels (DeviceHandle *deviceHandle*, Int32 * *numChannels*)

Get the number of Digital Input channels

Parameters:

deviceHandle The handle to an existing device

numChannels The returned number of Digital Input channels

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.274 UeiDaqAPI int UeiDaqGetDeviceNumberOfDOChannels (DeviceHandle *deviceHandle*, Int32 * *numChannels*)

Get the number of Digital Output channels

Parameters:

deviceHandle The handle to an existing device

numChannels The returned number of Digital Output channels

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.275 UeiDaqAPI int UeiDaqGetDeviceNumberOfSerialPorts (DeviceHandle *deviceHandle*, Int32 * *numSerialPorts*)

Get the number of serial ports

Parameters:

deviceHandle The handle to an existing device

numSerialPorts The returned number of serial ports

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.276 UeiDaqAPI int UeiDaqGetDeviceNumberOfSynchronousSerialPorts (DeviceHandle *deviceHandle*, Int32 * *numSynchronousSerialPorts*)

Get the number of synchronous serial ports

Parameters:

deviceHandle The handle to an existing device

numSynchronousSerialPorts The returned number of serial ports

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.277 UeiDaqAPI int UeiDaqGetDeviceOutputFIFOSize (DeviceHandle *deviceHandle*, Int32 * *outputFIFOSize*)

Get the size of the output FIFO in number of samples

Parameters:

deviceHandle The handle to an existing device

outputFIFOSize the number of samples that can be stored in the output FIFO

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.278 UeiDaqAPI int UeiDaqGetDeviceSerialNumber (DeviceHandle *deviceHandle*, char * *serialNumber*, int * *serialNumberLength*)

Get the serial number from the device EEPROM

Parameters:

deviceHandle The handle to an existing device

serialNumber A string to contain the returned device serial number.

serialNumberLength The length of the string, on function return contains the number of characters copied. If serialNumber is NULL, serialNumberLength contains the required length for serialNumber

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.279 UeiDaqAPI int UeiDaqGetDeviceStatus (DeviceHandle *deviceHandle*, char * *status*, int * *statusLength*)

Get a string containing low-level information about the device status

Parameters:

deviceHandle The handle to an existing device

status the hardware information string

statusLength The length of the info string, on function return contains the number of characters copied. If info is NULL, infoLength contains the required length for info

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.280 UeiDaqAPI int UeiDaqGetDeviceSupportsDigitalFiltering (DeviceHandle *deviceHandle*, Int32 * *digitalFiltering*)

Check if the device supports digital filtering on its Counter/timer inputs

Parameters:

deviceHandle The handle to an existing device

digitalFiltering set to 1 if the device has digital FIR filters

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.281 UeiDaqAPI int UeiDaqGetDeviceTimeout (DeviceHandle *deviceHandle*, Int32 * *pTimeout*)

Get the maximum amount of time (in ms) for a low-level access request to complete.

Parameters:

deviceHandle The handle to an existing device

pTimeout the timeout

Returns:

The status code

See also:

UeiDaqSetDeviceTimeout (p. 915), **UeiDaqGetDeviceHandle** (p. 795)

3.2.3.282 UeiDaqAPI int UeiDaqGetDIChannelEdgeMask (ChannelHandle *channelHandle*, uInt32 * *mask*)

Get the mask used to select wich digital line will be used to sense edges. When bit x is set to 1, line x is used.

Parameters:

channelHandle The handle to an existing channel

mask The returned edge detection mask.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.283 UeiDaqAPI int UeiDaqGetDIIndustrialHighThreshold (ChannelHandle *channelHandle*, int *line*, double * *pHighThreshold*)

The high input threshold is used in combination with the low input threshold to define an hysteresis window. The input line state will change only when the input signal crosses both threshold.

Parameters:

channelHandle The handle to an existing industrial DI channel

line The input line to query or configure within the port

pHighThreshold the current high threshold

Returns:

The status code.

See also:

UeiDaqSetDIIndustrialHighThreshold (p. 916)

3.2.3.284 UeiDaqAPI int UeiDaqGetDIIndustrialInputGain (ChannelHandle *channelHandle*, int *line*, int * *pInputGain*)

Some DI devices use an ADC to measure the input line state. This setting provides a way to select the input gain for the ADC. The input gain value is an index in the gain list starting with 0

Parameters:

channelHandle The handle to an existing industrial DI channel

line The input line to query

pInputGain the line's input gain

Returns:

The status code.

3.2.3.285 UeiDaqAPI int UeiDaqGetDIIndustrialLowThreshold (ChannelHandle *channelHandle*, int *line*, double * *pLowThreshold*)

The low input threshold is used in combination with the high input threshold to define an hysteresis window. The input line state will change only when the input signal crosses both threshold.

Parameters:

channelHandle The handle to an existing industrial DI channel

line The input line to query or configure within the port

pLowThreshold the current low threshold

Returns:

The status code.

See also:

UeiDaqSetDIIndustrialLowThreshold (p. 917)

3.2.3.286 UeiDaqAPI int UeiDaqGetDIIndustrialMinimumPulseWidth (ChannelHandle *channelHandle*, int *line*, double * *pMinWidth*)

The digital input filter is used to block digital noise from switching the input line state. It filters glitches or spikes based on their width.

Parameters:

channelHandle The handle to an existing DO protected channel

line The input line to query or configure within the port

pMinWidth the current minimum pulse width in ms

Returns:

The status code.

See also:

UeiDaqSetDIIndustrialMinimumPulseWidth (p. 917)

3.2.3.287 UeiDaqAPI int UeiDaqGetDIIndustrialMux (ChannelHandle *channelHandle*, int *line*, tUeiDigitalInputMux * *pMux*)

Select how voltage supply is connected to each input line. Disconnected: Set the mux to tri-state mode to disconnect voltage supply from input line Diag mode: Set mux to Diagnostic mode to connect internal voltage source to each input line in order to test that it is functional Pull-up mode: Setting mux to pull-up mode connects a pull-up resistor between the internal voltage source and the input line to monitor switches or contacts without external circuitry.

Parameters:

channelHandle The handle to an existing industrial DI channel

line The input line to query
pMux the current mux state

Returns:

The status code.

3.2.3.288 UeiDaqAPI int UeiDaqGetDIIndustrialMuxDelay (ChannelHandle *channelHandle*, int *line*, int * *pMuxDelayUs*)

Some DI devices use an ADC to measure the input line state. This setting provides a way to set the delay for the multiplexer in front of the ADC

Parameters:

channelHandle The handle to an existing industrial DI channel
line The input line to query
pMuxDelayUs the mux delay in us

Returns:

The status code.

3.2.3.289 UeiDaqAPI double UeiDaqGetDIIndustrialVoltageSupply (ChannelHandle *channelHandle*, int *line*, double * *pVoltage*)

Get the voltage supplied to selected line. Test mode: Guardian feature that connects an internal voltage source to each input line in order to test that it is functional Pull-up mode: connect a pull-up resistor between the internal voltage source and the input line to monitor switch or contacts without external circuitry.

Parameters:

channelHandle The handle to an existing industrial DI channel
line The input line to query
pVoltage the test voltage

Returns:

The status code.

3.2.3.290 UeiDaqAPI int UeiDaqGetDMMFIRCutoff (ChannelHandle *channelHandle*, tUeiDMMFIRCutoff * *frequency*)

Get the FIR cutoff frequency

Parameters:

channelHandle The handle to a DMM channel
frequency The current cutoff frequency setting

Returns:

The status code

3.2.3.291 UeiDaqAPI int UeiDaqGetDOChannelDefaultValue (ChannelHandle *channelHandle*, uInt32 * *pDefaultVal*)

Get the default value.

Parameters:

channelHandle The handle to an existing channel

pDefaultVal the default value.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.292 UeiDaqAPI int UeiDaqGetDOProtectedAutoRetry (ChannelHandle *channelHandle*, int * *pAutoRetry*)

The auto retry setting specifies wheter the device should attempt to close the circuit after it was opened because of an over or under current condition. If it is set to true the device will try to close the circuit at a rate set by the autoRetryRate setting.

Parameters:

channelHandle The handle to an existing DO protected channel

pAutoRetry true if autoRetry is on, false otherwise.

Returns:

The status code.

See also:

UeiDaqSetDOProtectedAutoRetry (p. 919)

3.2.3.293 UeiDaqAPI int UeiDaqGetDOProtectedAutoRetryRate (ChannelHandle *channelHandle*, double * *pAutoRetryRate*)

Specifies how often the device will attemp to close a circuit that was opened because of an over or under current condition.

Parameters:

channelHandle The handle to an existing DO protected channel

pAutoRetryRate The number of retries per second.

Returns:

The status code.

See also:

UeiDaqSetDOProtectedAutoRetryRate (p. 920)

3.2.3.294 UeiDaqAPI int UeiDaqGetDOProtectedCurrentMeasurementRate (ChannelHandle *channelHandle*, double * *pMeasurementRate*)

Get the rate at which the current is measured. This rate determines how fast the device react when an under or over current condition occurs.

Parameters:

channelHandle The handle to an existing DO protected channel

pMeasurementRate The current measurement rate.

Returns:

The status code.

See also:

UeiDaqGetDOProtectedCurrentMeasurementRate (p. 810)

3.2.3.295 UeiDaqAPI int UeiDaqGetDOProtectedOverCurrentLimit (ChannelHandle *channelHandle*, int *line*, double * *pOverCurrent*)

Get the maximum amount of current allowed in Amps. If more than the maximum current flows through the digital output line, the circuit will open.

Parameters:

channelHandle The handle to an existing DO protected channel

line The output line to configure within the port

pOverCurrent The over current limit.

Returns:

The status code.

See also:

UeiDaqSetDOProtectedUnderCurrentLimit (p. 922) **UeiDaqGetDOProtectedUnderCurrentLimit** (p. 812) **UeiDaqSetDOProtectedOverCurrentLimit** (p. 920)

3.2.3.296 UeiDaqAPI int UeiDaqGetDOProtectedOverUnderCount (ChannelHandle *channelHandle*, uInt32 * *pOverUnderCount*)

Specifies number of consecutive over/under limit diagnostic readings that must occur in order to trip breaker.

Parameters:

channelHandle The handle to an existing channel

pOverUnderCount The maximum number of over/under readings.

Returns:

The status code

3.2.3.297 UeiDaqAPI int UeiDaqGetDOProtectedPWMDutyCycle (ChannelHandle *channelHandle*, int *line*, double * *pDutyCycle*)

Specifies the continuous pulse train duty cycle

Parameters:

channelHandle The handle to an existing DO protected channel
line The output line to configure within the port
pDutyCycle The current pulse train period.

Returns:

The status code.

See also:

UeiDaqSetDOProtectedPWMDutyCycle (p. 921)

3.2.3.298 UeiDaqAPI int UeiDaqGetDOProtectedPWMLength (ChannelHandle *channelHandle*, int *line*, uInt32 * *pLengthus*)

Specifies soft start/stop pulse train length in micro-seconds

Parameters:

channelHandle The handle to an existing DO protected channel
line The output line to configure within the port
pLengthus The current pulse train length.

Returns:

The status code.

See also:

UeiDaqSetDOProtectedPWMLength (p. 921)

3.2.3.299 UeiDaqAPI int UeiDaqGetDOProtectedPWMMode (ChannelHandle *channelHandle*, int *line*, tUeiDOPWMMode * *pMode*)

Specifies the PWM mode for DO channels that support the capability With PWM mode you can replace the rising and falling edges with a pulse train to allow for soft start and/or stop

Parameters:

channelHandle The handle to an existing DO protected channel
line The output line to configure within the port
pMode The current PWM mode.

Returns:

The status code.

See also:

UeiDaqSetDOProtectedPWMMode (p. 922)

3.2.3.300 UeiDaqAPI int UeiDaqGetDOProtectedPWMPeriod (ChannelHandle *channelHandle*, int *line*, uInt32 * *pPeriodus*)

Specifies soft start/stop or continuous pulse train period in micro-seconds

Parameters:

channelHandle The handle to an existing DO protected channel

line The output line to configure within the port

pPeriodus The current pulse train period.

Returns:

The status code.

See also:

UeiDaqSetDOProtectedPWMPeriod (p. 922)

3.2.3.301 UeiDaqAPI int UeiDaqGetDOProtectedUnderCurrentLimit (ChannelHandle *channelHandle*, int *line*, double * *pUnderCurrent*)

Get the minimum amount of current allowed in Amps. If less than the minimum current flows through the digital output line, the circuit will open.

Parameters:

channelHandle The handle to an existing DO protected channel

line The output line to configure within the port

pUnderCurrent The under current limit.

Returns:

The status code.

See also:

UeiDaqSetDOProtectedUnderCurrentLimit (p. 922) **UeiDaqGetDOProtectedOverCurrentLimit** (p. 810) **UeiDaqSetDOProtectedOverCurrentLimit** (p. 920)

3.2.3.302 UeiDaqAPI int UeiDaqGetFrameworkInstallationDirectory (char * *installDir*, int *installDirSize*)

Get the framework installation directory

Parameters:

installDir The string that will contain the installation directory once the function returns

installDirSize The size of the string

Returns:

The status code.

3.2.3.303 UeiDaqAPI int UeiDaqGetFrameworkVersion (char * *version*, int *versionSize*)

Get the the four parts version string. "major.minor.extra.build"

Parameters:

version The string that will contain the version once the function returns
versionSize The size of the version string

Returns:

The status code.

3.2.3.304 UeiDaqAPI int UeiDaqGetI2CCustomSpeed (ChannelHandle *channelHandle*, float * *bitRate*)

Get the I2C bus custom bit rate

Parameters:

channelHandle The handle to an I2C slave or master port
bitRate The current I2C bus custom bit rate

Returns:

The status code

3.2.3.305 UeiDaqAPI int UeiDaqGetI2CMasterByteToByteDelay (ChannelHandle *channelHandle*, uInt16 * *delayUs*)

Get the delay in microseconds that the master will delay between sending bytes

Parameters:

channelHandle The handle to an I2C master port
delayUs the current delay in microseconds

Returns:

The status code

3.2.3.306 UeiDaqAPI int UeiDaqGetI2CMasterLoopbackMode (ChannelHandle *channelHandle*, tUeiI2CLoopback * *mode*)

Get the I2C loopback setting

Parameters:

channelHandle The handle to an I2C master port
mode The current I2C loopback mode

Returns:

The status code

3.2.3.307 UeiDaqAPI int UeiDaqGetI2CMasterMaxClockStretchingDelay (ChannelHandle *channelHandle*, uInt16 * *delayUs*)

Get the time in microseconds that the master will allow a slave to delay the clock

Parameters:

channelHandle The handle to an I2C master port

delayUs current the time in microseconds

Returns:

The status code

3.2.3.308 UeiDaqAPI int UeiDaqGetI2CSlaveAddress (ChannelHandle *channelHandle*, int * *address*)

Get the I2C slave address

Parameters:

channelHandle The handle to an I2C slave port

address The current I2C slave address

Returns:

The status code

3.2.3.309 UeiDaqAPI int UeiDaqGetI2CSlaveAddressWidth (ChannelHandle *channelHandle*, tUeiI2CSlaveAddressWidth * *width*)

Get the I2C slave address width

Parameters:

channelHandle The handle to an I2C slave port

width The current I2C slave address width

Returns:

The status code

3.2.3.310 UeiDaqAPI int UeiDaqGetI2CSlaveClockStretchingDelay (ChannelHandle *channelHandle*, uInt16 * *clocks*)

Get the delay in 15ns clock increments that the slave will delay an ACK. The clock stretching delay must also be individually enabled for address, transmit, and receive cycles. Max clock stretching delay is ~62ms.

Parameters:

channelHandle The handle to an I2C master port

clocks the delay in clocks

Returns:

The status code

3.2.3.311 UeiDaqAPI int UeiDaqGetI2CSlaveMaxWordsPerAck (ChannelHandle *channelHandle*, int * *maxWords*)

Get maximum number of words slave will receive per transmission

Parameters:

channelHandle The handle to an I2C master port

maxWords the maximum number of words slave will receive per transmission

Returns:

The status code

3.2.3.312 UeiDaqAPI int UeiDaqGetI2CSlaveRegisterData (ChannelHandle *channelHandle*, int *byteIndex*, uInt8 * *data*)

Get data transmitted when in Register mode or padding data when in FIFO mode.

Parameters:

channelHandle The handle to an I2C slave port

byteIndex Index of the byte to get, 0-3

data The current byte data

Returns:

The status code

3.2.3.313 UeiDaqAPI int UeiDaqGetI2CSlaveRegisterDataSize (ChannelHandle *channelHandle*, int * *size*)

Get the number of bytes used in slave data when in Register data mode.

Parameters:

channelHandle The handle to an I2C master port

size the number of bytes used in slave data when in Register data mode

Returns:

The status code

3.2.3.314 UeiDaqAPI int UeiDaqGetI2CSlaveTransmitDataMode (ChannelHandle *channelHandle*, tUeiI2CSlaveDataMode * *dataMode*)

Get slave transmit data mode

Parameters:

channelHandle The handle to an I2C slave port

dataMode the slave transmit data mode

Returns:

The status code

3.2.3.315 UeiDaqAPI int UeiDaqGetI2CSpeed (ChannelHandle *channelHandle*, tUeiI2CPortSpeed * *bitRate*)

Get the I2C bus bit rate

Parameters:

channelHandle The handle to an I2C slave or master port

bitRate The current I2C bus bit rate

Returns:

The status code

3.2.3.316 UeiDaqAPI int UeiDaqGetI2CTTLLevel (ChannelHandle *channelHandle*, tUeiI2CTTLLevel * *ttlLevel*)

Get the I2C bus TTL level

Parameters:

channelHandle The handle to an I2C slave or master port

ttlLevel The current I2C bus TTL level

Returns:

The status code

3.2.3.317 UeiDaqAPI int UeiDaqGetIRIGInputTimeCodeFormat (ChannelHandle *channelHandle*, tUeiIRIGTimeCodeFormat * *format*)

Get the time code format used by the signal connected to the device.

Parameters:

channelHandle The handle to an existing IRIG channel

format the time code format.

Returns:

The status code

See also:

UeiDaqSetIRIGTimeCodeFormat

3.2.3.318 UeiDaqAPI int UeiDaqGetIRIGInputTimeDecoderInput (ChannelHandle *channelHandle*, tUeiIRIGDecoderInputType * *input*)

Get the type of time code connected to the device.

Parameters:

channelHandle The handle to an existing IRIG channel

input the time decoder input type.

Returns:

The status code

See also:

UeiDaqSetIRIGTimeDecoderInput

3.2.3.319 UeiDaqAPI int UeiDaqGetIRIGTimeKeeper1PPSSource (ChannelHandle *channelHandle*, tUeiIRIGTimeKeeper1PPSSource * *source*)

For proper functioning, the timekeeper requires a 1PPS signal. which can be delivered from multiple sources: internal, external, GPS... This method gets the current time keeper 1PPS source

Parameters:

channelHandle The handle to an existing MIL-1553 channel

source the 1PPS source.

Returns:

The status code

See also:

UeiDaqSetIRIG1PPSSource

3.2.3.320 UeiDaqAPI int UeiDaqGetLVDTExcitationFrequency (ChannelHandle *channelHandle*, double * *pFex*)

Get the excitation frequency configured for the channel.

Parameters:

channelHandle The handle to an existing LVDT channel

pFex The excitation frequency

Returns:

The status code.

See also:

UeiDaqSetLVDTExcitationFrequency (p. 927)

3.2.3.321 UeiDaqAPI int UeiDaqGetLVDTExcitationVoltage (ChannelHandle *channelHandle*, double * *pVex*)

Get the excitation RMS voltage configured for the channel.

Parameters:

channelHandle The handle to an existing LVDT channel

pVex The excitation voltage

Returns:

The status code.

See also:

UeiDaqSetLVDTExcitationVoltage (p. 928)

3.2.3.322 UeiDaqAPI int UeiDaqGetLVDTSensorSensitivity (ChannelHandle *channelHandle*, double * *pSensitivity*)

Get the sensor sensitivity, it usually is in mVolts/V/mm Independently of the unit, the voltage read will be converted to mV and divided by the excitation voltage and the sensitivity

Parameters:

channelHandle The handle to an existing LVDT channel

pSensitivity The current sensitivity

Returns:

The status code.

See also:

UeiDaqSetLVDTSensorSensitivity (p. 928)

3.2.3.323 UeiDaqAPI int UeiDaqGetLVDTWiringScheme (ChannelHandle *channelHandle*, tUeiLVDTWiringScheme * *pWiring*)

Get the current wiring scheme used for this channel.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

pWiring The current wiring scheme.

Returns:

The status code.

See also:

UeiDaqSetLVDTWiringScheme (p. 928)

3.2.3.324 UeiDaqAPI int UeiDaqGetMIL1553Coupling (ChannelHandle *channelHandle*, tUeiMIL1553PortCoupling * *pCoupling*)

Get the coupling used to to connect the port to the bus

Parameters:

channelHandle The handle to an existing MIL-1553 channel

pCoupling The 1553 port coupling.

Returns:

The status code.

See also:

UeiDaqSetMIL1553Coupling (p. 929)

3.2.3.325 UeiDaqAPI int UeiDaqGetMIL1553FilterEntry (ChannelHandle *channelHandle*, int *index*, tUeiMIL1553FilterEntry ** *entry*)

Retrieve a filter entry from the port's frame filter table. Returns NULL when retrieving past the end of the table.

Parameters:

channelHandle The handle to an existing MIL-1553 channel

index The index of the filter entry to retrieve

entry A pointer to the retrieved filter entry

Returns:

The status code.

See also:

UeiDaqAddMIL1553FilterEntry (p. 706), **UeiDaqClearMIL1553FilterEntries** (p. 708)

3.2.3.326 UeiDaqAPI int UeiDaqGetMIL1553PortMode (ChannelHandle *channelHandle*, tUeiMIL1553PortOpMode * *pPortMode*)

Get the port mode of operation: BM, RT or BC

Parameters:

channelHandle The handle to an existing MIL-1553 channel

pPortMode The 1553 port mode of operation

Returns:

The status code.

See also:

UeiDaqSetMIL1553PortMode (p. 929)

3.2.3.327 UeiDaqAPI int UeiDaqGetMIL1553RxBus (ChannelHandle *channelHandle*, tUeiMIL1553PortActiveBus * *pPortBus*)

Get the active bus: A or B or both

Parameters:

channelHandle The handle to an existing MIL-1553 channel

pPortBus The 1553 port active bus for reception

Returns:

The status code.

See also:

UeiDaqSetMIL1553RxBus (p. 930)

3.2.3.328 UeiDaqAPI int UeiDaqGetMIL1553SchedulerEntry (ChannelHandle *channelHandle*, int *index*, tUeiMIL1553SchedulerEntry ** *entry*)

Retrieve a scheduler entry from the port's scheduler table. Returns NULL when retrieving past the end of the table.

Parameters:

channelHandle The handle to an existing MIL-1553 channel

index The index of the scheduler entry to retrieve

entry A pointer to the retrieved scheduler entry

Returns:

The status code.

See also:

UeiDaqAddMIL1553SchedulerEntry (p. 706), **UeiDaqClearMIL1553SchedulerEntries** (p. 709)

3.2.3.329 UeiDaqAPI int UeiDaqGetMIL1553TxBus (ChannelHandle *channelHandle*, tUeiMIL1553PortActiveBus * *pPortBus*)

Get the active bus: A or B

Parameters:

channelHandle The handle to an existing MIL-1553 channel

pPortBus The 1553 port active bus for transmission

Returns:

The status code.

See also:

UeiDaqSetMIL1553TxBus (p. 930)

3.2.3.330 UeiDaqAPI int UeiDaqGetMuxOffDelay (ChannelHandle *channelHandle*, int * *delay*)

Get the current port off delay in microseconds

Parameters:

channelHandle The handle to a MUX port

delay the current off delay

Returns:

The status code

3.2.3.331 UeiDaqAPI int UeiDaqGetMuxOnDelay (ChannelHandle *channelHandle*, int * *delay*)

Get the current port on delay in microseconds

Parameters:

channelHandle The handle to a MUX port

delay the current on delay

Returns:

The status code

3.2.3.332 UeiDaqAPI int UeiDaqGetMuxSyncInputEdgePolarity (ChannelHandle *channelHandle*, tUeiDigitalEdge * *polarity*)

Get the current port synchronization input edge/level polarity

Parameters:

channelHandle The handle to a MUX port

polarity the current synchronization input polarity

Returns:

The status code

3.2.3.333 UeiDaqAPI int UeiDaqGetMuxSyncOutputMode (ChannelHandle *channelHandle*, tUeiMuxSyncOutputMode * *mode*)

Get the current port synchronization output mode A MUX device can emit a pulse on its synchronization output when configuring its relays

Parameters:

channelHandle The handle to a MUX port

mode the current synchronization output mode

Returns:

The status code

3.2.3.334 UeiDaqAPI int UeiDaqGetMuxSyncOutputPulseWidth (ChannelHandle *channelHandle*, int * *width*)

Get the current port synchronization output pulse width in microseconds

Parameters:

channelHandle The handle to a MUX port

width the current synchronization output pulse width

Returns:

The status code

3.2.3.335 UeiDaqAPI int UeiDaqGetNumberOfChannels (SessionHandle *sessionHandle*, int * *numChannels*)

Get the number of channels in the channel list.

Parameters:

sessionHandle The handle to an existing session

numChannels The number of channels.

Returns:

The status code

3.2.3.336 UeiDaqAPI int UeiDaqGetNumberOfPreTriggerScans (TriggerHandle *triggerHandle*, Int32 * *pNumPreTriggerScans*)

The number of scans saved before the trigger occurred.

Parameters:

triggerHandle The handle to an existing trigger object
pNumPreTriggerScans the number of pre-trigger scans

Returns:

The status code

See also:

UeiDaqSetNumberOfPreTriggerScans (p. 932) **UeiDaqGetStartTriggerHandle** (p. 833) **UeiDaqGetStopTriggerHandle** (p. 833)

3.2.3.337 UeiDaqAPI tUeiOSType UeiDaqGetOperatingSystemType ()

Get the type of the operating system installed on this computer

Returns:

The operating system type.

3.2.3.338 UeiDaqAPI int UeiDaqGetPluginLowLevelDriverVersion (char * *driverName*, char * *version*, int *versionSize*)

Get the the four parts version string of the specified driver plugin. "major.minor.extra.build"

Parameters:

driverName The name of the driver plugin
version The string that will contain the version once the function returns
versionSize The size of the version string

Returns:

The status code.

3.2.3.339 UeiDaqAPI int UeiDaqGetPluginsInstallationDirectory (char * *installDir*, int *installDirSize*)

Get the framework plugins directory

Parameters:

installDir The string that will contain the installation directory once the function returns
installDirSize The size of the string

Returns:

The status code.

3.2.3.340 UeiDaqAPI int UeiDaqGetResistanceExcitationVoltage (ChannelHandle *channelHandle*, double * *pVex*)

Get the excitation voltage configured for the channel.

Parameters:

channelHandle The handle to an existing resistance channel

pVex The excitation voltage

Returns:

The status code.

See also:

UeiDaqSetResistanceExcitationVoltage (p. 932)

3.2.3.341 UeiDaqAPI int UeiDaqGetResistanceReferenceResistance (ChannelHandle *channelHandle*, double * *pRref*)

Get the reference resistance used to measure the current flowing through the resistive sensor.

Parameters:

channelHandle The handle to an existing resistance channel

pRref The current reference resistance.

Returns:

The status code.

See also:

UeiDaqSetResistanceReferenceResistance (p. 932)

3.2.3.342 UeiDaqAPI int UeiDaqGetResistanceReferenceResistorType (ChannelHandle *channelHandle*, tUeiReferenceResistorType * *pRefResistorType*)

The reference resistor can be built-in the connector block if you are using a DNA-STP-AIU or connected externally.

Parameters:

channelHandle The handle to an existing resistance channel

pRefResistorType The current reference resistor type.

Returns:

The status code.

See also:

UeiDaqSetResistanceReferenceResistorType (p. 933)

3.2.3.343 UeiDaqAPI int UeiDaqGetResistanceWiringScheme (ChannelHandle *channelHandle*, tUeiWiringScheme * *pWiring*)

Get the current wiring scheme used for this channel.

Parameters:

channelHandle The handle to an existing resistance channel

pWiring The current wiring scheme.

Returns:

The status code.

See also:

UeiDaqSetResistanceWiringScheme (p. 933)

3.2.3.344 UeiDaqAPI int UeiDaqGetRTDCoefficientA (ChannelHandle *channelHandle*, double * *pA*)

The Callendar-Van Dusen equation is an equation that describes the relationship between resistance (R) and temperature (t) of platinum resistance thermometers. t < 0: R(t) = R(0)[1 + A * t + B * t * t + (t - 100)C * t * t * t]. t > 0: R(t) = R(0)(1 + A * t + B * t * t).

Parameters:

channelHandle The handle to an existing resistance channel

pA The current CVD A coefficient.

Returns:

The status code.

See also:

UeiDaqSetRTDCoefficientA (p. 933)

3.2.3.345 UeiDaqAPI int UeiDaqGetRTDCoefficientB (ChannelHandle *channelHandle*, double * *pB*)

The Callendar-Van Dusen equation is an equation that describes the relationship between resistance (R) and temperature (t) of platinum resistance thermometers. t < 0: R(t) = R(0)[1 + A * t + B * t * t + (t - 100)C * t * t * t]. t > 0: R(t) = R(0)(1 + A * t + B * t * t).

Parameters:

channelHandle The handle to an existing resistance channel

pB The current CVD B coefficient.

Returns:

The status code.

See also:

UeiDaqSetRTDCoefficientB (p. 934)

3.2.3.346 UeiDaqAPI int UeiDaqGetRTDCoefficientC (ChannelHandle *channelHandle*, double * *pC*)

The Callendar-Van Dusen equation is an equation that describes the relationship between resistance (R) and temperature (t) of platinum resistance thermometers. t < 0: R(t) = R(0)[1 + A * t + B * t * t + (t - 100)C * t * t * t]. t > 0: R(t) = R(0)(1 + A * t + B * t * t).

Parameters:

channelHandle The handle to an existing resistance channel

pC The current CVD C coefficient.

Returns:

The status code.

See also:

UeiDaqSetRTDCoefficientC (p. 934)

3.2.3.347 UeiDaqAPI int UeiDaqGetRTDNominalResistance (ChannelHandle *channelHandle*, double * *pR0*)

Get the RTD nominal resistance at 0 deg.

Parameters:

channelHandle The handle to an existing resistance channel

pR0 The current RTD nominal resistance.

Returns:

The status code.

See also:

UeiDaqSetRTDNominalResistance (p. 935)

3.2.3.348 UeiDaqAPI int UeiDaqGetRTDTemperatureScale (ChannelHandle *channelHandle*, tUeiTemperatureScale * *pTempScale*)

Get the temperature scale used to convert the measured resistance to temperature.

Parameters:

channelHandle The handle to an existing RTD channel

pTempScale the actual temperature scale.

Returns:

The status code.

See also:

UeiDaqSetTCTemperatureScale (p. 943)

3.2.3.349 UeiDaqAPI int UeiDaqGetRTDType (ChannelHandle *channelHandle*, tUeiRTDType * *pRTDType*)

RTD sensors are specified using the "alpha" (a) constant It is also known as the temperature coefficient of resistance, and symbolizes the resistance change factor per degree of temperature change. The RTD type is used to select the proper coefficients A, B and C for the Callendar Van-Dusen equation used to convert resistance measurements to temperature.

Parameters:

channelHandle The handle to an existing resistance channel

pRTDType The current RTD type.

Returns:

The status code.

See also:

UeiDaqSetRTDType (p. 935)

3.2.3.350 UeiDaqAPI int UeiDaqGetSerialPortCustomSpeed (ChannelHandle *channelHandle*, uInt32 * *pBitsPerSecond*)

Get the number of data bits transmitted per second. Call this function when using a non-standard port speed

Parameters:

channelHandle The handle to an existing serial port

pBitsPerSecond the serial port speed .

Returns:

The status code.

See also:

UeiDaqSetSerialPortCustomSpeed (p. 936)

3.2.3.351 UeiDaqAPI int UeiDaqGetSerialPortDataBits (ChannelHandle *channelHandle*, tUeiSerialPortDataBits * *pDataBits*)

Get the number of data bits that hold the data to be transferred.

Parameters:

channelHandle The handle to an existing serial port

pDataBits The number of data bits.

Returns:

The status code.

See also:

UeiDaqSetSerialPortDataBits (p. 936)

3.2.3.352 UeiDaqAPI int UeiDaqGetSerialPortFlowControl (ChannelHandle *channelHandle*, tUeiSerialPortFlowControl * *pFlowControl*)

Get flow control setting

Parameters:

channelHandle The handle to an existing serial port

pFlowControl The current flow control setting

Returns:

The status code.

See also:

UeiDaqSetSerialPortFlowControl (p. 936)

3.2.3.353 UeiDaqAPI int UeiDaqGetSerialPortMode (ChannelHandle *channelHandle*, tUeiSerialPortMode * *pMode*)

Get the mode used by the serial port. Possible modes are RS-232, RS-485 half duplex and RS-485 full duplex

Parameters:

channelHandle The handle to an existing serial port

pMode the serial port mode.

Returns:

The status code.

See also:

UeiDaqSetSerialPortMode (p. 937)

3.2.3.354 UeiDaqAPI int UeiDaqGetSerialPortParity (ChannelHandle *channelHandle*, tUeiSerialPortParity * *pParity*)

Get the parity used to detect transmission errors.

Parameters:

channelHandle The handle to an existing serial port

pParity the parity.

Returns:

The status code.

See also:

UeiDaqSetSerialPortParity (p. 937)

3.2.3.355 UeiDaqAPI int UeiDaqGetSerialPortSpeed (ChannelHandle *channelHandle*, tUeiSerialPortSpeed * *pBitsPerSecond*)

Get the number of data bits transmitted per second.

Parameters:

channelHandle The handle to an existing serial port

pBitsPerSecond the serial port speed .

Returns:

The status code.

See also:

UeiDaqSetSerialPortSpeed (p. 937)

3.2.3.356 UeiDaqAPI int UeiDaqGetSerialPortTermination (ChannelHandle *channelHandle*, char * *pEol*, int * *eolLength*)

Get the sequence of characters used to define the end of a line.

Parameters:

channelHandle The handle to an existing serial port

pEol The termination characters

eolLength The maximum length of the termination string

Returns:

The status code.

See also:

UeiDaqSetSerialPortTermination (p. 938)

3.2.3.357 UeiDaqAPI int UeiDaqGetSessionCustomProperty (SessionHandle *sessionHandle*, char * *property*, int *valueSize*, void * *value*)

Get a custom property on the device or subsystem associated with this session Custom properties are device dependent, refer to the device user manual to learn about the properties supported by your device

Parameters:

sessionHandle The handle of the session

property string containing the name of the property to get

valueSize size in bytes of the memory block receiving the property value

value pointer to the memory block receiving the property value

Returns:

The status code

3.2.3.358 UeiDaqAPI int UeiDaqGetSessionGroupDirectory (char * *sessionGroup*, char * *sessionGroupDir*, int *sessionGroupDirSize*)

Get a session group directory

Parameters:

sessionGroup the name of the session group

sessionGroupDir The string that will contain the installation directory once the function returns

sessionGroupDirSize The size of the string

Returns:

The status code.

3.2.3.359 UeiDaqAPI int UeiDaqGetSessionType (SessionHandle *sessionHandle*, tUeiSessionType * *sessionType*)

Get the session type of this session. The session type is set by the first channel added to the session.

Parameters:

sessionHandle The handle of the session

sessionType The type of the session

Returns:

The status code

3.2.3.360 UeiDaqAPI int UeiDaqGetSimulatedLVDTExcitationFrequency (ChannelHandle *channelHandle*, double * *pFex*)

Get the excitation frequency configured for the channel.

Parameters:

channelHandle The handle to an existing simulated LVDT channel

pFex The excitation frequency

Returns:

The status code.

See also:

UeiDaqSetSimulatedLVDTExcitationFrequency (p. 939)

3.2.3.361 UeiDaqAPI int UeiDaqGetSimulatedLVDTExcitationVoltage (ChannelHandle *channelHandle*, double * *pVex*)

Get the excitation RMS voltage configured for the channel.

Parameters:

channelHandle The handle to an existing simulated LVDT channel

pVex The excitation voltage

Returns:

The status code.

See also:

UeiDaqSetSimulatedLVDTExcitationVoltage (p. 939)

3.2.3.362 UeiDaqAPI int UeiDaqGetSimulatedLVDTSensorSensitivity (ChannelHandle *channelHandle*, double * *pSensitivity*)

Get the sensor sensitivity, it usually is in mVolts/V/mm Independently of the unit, the voltage read will be converted to mV and divided by the excitation voltage and the sensitivity

Parameters:

channelHandle The handle to an existing simulated LVDT channel

pSensitivity The current sensitivity

Returns:

The status code.

See also:

UeiDaqSetSimulatedLVDTSensorSensitivity (p. 940)

3.2.3.363 UeiDaqAPI int UeiDaqGetSimulatedLVDTWiringScheme (ChannelHandle *channelHandle*, tUeiLVDTWiringScheme * *pWiring*)

Get the current wiring scheme used for this channel.

Parameters:

channelHandle The handle to an existing simulated LVDT channel

pWiring The current wiring scheme.

Returns:

The status code.

See also:

UeiDaqSetSimulatedLVDTWiringScheme (p. 940)

3.2.3.364 UeiDaqAPI int UeiDaqGetSimulatedSynchroResolverExcitationFrequency (ChannelHandle *channelHandle*, double * *pFex*)

Get the excitation frequency configured for the channel.

Parameters:

channelHandle The handle to an existing simulated synchro/resolver channel

pFex The excitation frequency.

Returns:

The status code.

See also:

UeiDaqSetSimulatedSynchroResolverExcitationFrequency (p. 940)

3.2.3.365 UeiDaqAPI int UeiDaqGetSimulatedSynchroResolverExcitationVoltage (ChannelHandle *channelHandle*, double * *pVex*)

Get the excitation RMS voltage configured for the channel.

Parameters:

channelHandle The handle to an existing simulated synchro/resolver channel

pVex The excitation voltage.

Returns:

The status code.

See also:

UeiDaqSetSimulatedSynchroResolverExcitationVoltage (p. 941)

3.2.3.366 UeiDaqAPI int UeiDaqGetSimulatedSynchroResolverMode (ChannelHandle *channelHandle*, tUeiSynchroResolverMode * *pMode*)

Specifies whether the sensor simulated is a synchro or a resolver.

Parameters:

channelHandle The handle to an existing simulated synchro/resolver channel

pMode The current simulated sensor type.

Returns:

The status code.

See also:

UeiDaqSetSimulatedSynchroResolverMode (p. 941)

3.2.3.367 UeiDaqAPI int UeiDaqGetStartTriggerHandle (SessionHandle *sessionHandle*, TriggerHandle * *trigger*)

Get a handle to the start trigger object associated with the session.

Parameters:

sessionHandle The handle to an existing session

trigger A handle to the Trigger object that is used to access triggers data.

Returns:

The status code

3.2.3.368 UeiDaqAPI int UeiDaqGetStopBits (ChannelHandle *channelHandle*, tUeiSerialPortStopBits * *pStopBits*)

Get the number of stop bits that indicate the end of a data message.

Parameters:

channelHandle The handle to an existing serial port

pStopBits the number of stop bits.

Returns:

The status code.

See also:

UeiDaqSetSerialPortStopBits (p. 938)

3.2.3.369 UeiDaqAPI int UeiDaqGetStopTriggerHandle (SessionHandle *sessionHandle*, TriggerHandle * *trigger*)

Get a handle to the stop trigger object associated with the session.

Parameters:

sessionHandle The handle to an existing session

trigger A handle to the Trigger object that is used to access triggers data.

Returns:

The status code

3.2.3.370 UeiDaqAPI int UeiDaqGetSynchroResolverExcitationFrequency (ChannelHandle *channelHandle*, double * *pFex*)

Get the excitation frequency configured for the channel.

Parameters:

channelHandle The handle to an existing synchro/resolver channel

pFex The excitation frequency.

Returns:

The status code.

See also:

UeiDaqSetSynchroResolverExcitationFrequency (p. 941)

3.2.3.371 UeiDaqAPI int UeiDaqGetSynchroResolverExcitationVoltage (ChannelHandle *channelHandle*, double * *pVex*)

Get the excitation RMS voltage configured for the channel.

Parameters:

channelHandle The handle to an existing synchro/resolver channel

pVex The excitation voltage.

Returns:

The status code.

See also:

UeiDaqSetSynchroResolverExcitationVoltage (p. 942)

3.2.3.372 UeiDaqAPI int UeiDaqGetSynchroResolverMode (ChannelHandle *channelHandle*, tUeiSynchroResolverMode * *pMode*)

Specifies whether the input sensor is a synchro or a resolver.

Parameters:

channelHandle The handle to an existing synchro/resolver channel

pMode The current input sensor type.

Returns:

The status code.

See also:

UeiDaqSetSynchroResolverMode (p. 942)

3.2.3.373 UeiDaqAPI int UeiDaqGetTCCJCConstant (ChannelHandle *channelHandle*, double * *pCJCConstant*)

Get the cold junction compensation temperature constant. This setting is only used when the CJC type is set to UeiCJCTypeConstant. The unit is the same as the configured temperature scale.

Parameters:

channelHandle The handle to an existing TC channel

pCJCConstant the actual cold junction compensation constant.

Returns:

The status code.

See also:

UeiDaqSetTCCJCConstant (p. 942)

3.2.3.374 UeiDaqAPI int UeiDaqGetTCCJCResource (ChannelHandle *channelHandle*, char * *cjcResource*, int * *cjcResourceLength*)

Get the resource string describing how to measure the cold junction temperature. This setting is only used when the CJC type is set to UeiCJCTypeResource. The CJC resource format is similar to the channel list format: [cjcSensorType]://[Channel(s)]?[Paramters] For example: to measure CJC from a linear IC sensor connected to channel 20 the resource string is "ic://20?K=0.00295" this will measure the voltage on channel 20 and compute the CJC temperature with the formula: degK = V / 0.00295

Parameters:

channelHandle The handle to an existing TC channel

cjcResource the actual cold junction compensation resource.

cjcResourceLength the cold junction compensation resource string length.

Returns:

The status code.

See also:

UeiDaqSetTCCJCResource (p. 943)

3.2.3.375 UeiDaqAPI int UeiDaqGetTCCJCType (ChannelHandle *channelHandle*, tUeiColdJunctionCompensationType * *pCJCType*)

Get the type of the cold junction compensation used to convert from volts to temperature.

Parameters:

channelHandle The handle to an existing TC channel

pCJCType the actual cold junction compensation type.

Returns:

The status code.

See also:

UeiDaqSetTCCJCType (p. 943)

3.2.3.376 UeiDaqAPI int UeiDaqGetTCTemperatureScale (ChannelHandle *channelHandle*, tUeiTemperatureScale * *pTempScale*)

Get the temperature scale used to convert the measured voltage to temperature.

Parameters:

channelHandle The handle to an existing TC channel

pTempScale the actual temperature scale.

Returns:

The status code.

See also:

UeiDaqSetTCTemperatureScale (p. 943)

3.2.3.377 UeiDaqAPI int UeiDaqGetTCThermocoupleType (ChannelHandle *channelHandle*, tUeiThermocoupleType * *pTCType*)

Get the type of the thermocouple connected to the channel.

Parameters:

channelHandle The handle to an existing TC channel

pTCType The actual thermocouple type

Returns:

The status code.

See also:

UeiDaqSetTCThermocoupleType (p. 944)

3.2.3.378 UeiDaqAPI int UeiDaqGetTimingConvertClockDestinationSignal (TimingHandle *timingHandle*, char * *signal*, int * *signalLength*)

Get the name of the signal driven by the conversion clock The signal name is device dependent.

Parameters:

timingHandle The handle to an existing timing object

signal The string to contain the current conversion clock destination signal

signalLength The length of the string, on function return contains the number of characters copied. If signal is NULL, signalLength contains the required length for signal

Returns:

the status code

See also:

UeiDaqSetTimingConvertClockDestinationSignal (p. 944)

3.2.3.379 UeiDaqAPI int UeiDaqGetTimingConvertClockEdge (TimingHandle *timingHandle*, tUeiDigitalEdge * *edge*)

Get the active edge of the conversion clock.

Parameters:

timingHandle The handle to an existing timing object

edge The returned active edge

Returns:

The status code

See also:

UeiDaqGetTimingHandle (p. 839)

3.2.3.380 UeiDaqAPI int UeiDaqGetTimingConvertClockRate (TimingHandle *timingHandle*, f64 * *rate*)

Get the rate of the conversion clock

Parameters:

timingHandle The handle to an existing timing object

rate The returned clock rate

Returns:

The status code

See also:

UeiDaqGetTimingHandle (p. 839)

3.2.3.381 UeiDaqAPI int UeiDaqGetTimingConvertClockSource (TimingHandle *timingHandle*, tUeiTimingClockSource * *source*)

Get the source of the A/D conversion clock

Parameters:

timingHandle The handle to an existing timing object

source The returned clock source

Returns:

The status code

See also:

UeiDaqGetTimingHandle (p. 839)

3.2.3.382 UeiDaqAPI int UeiDaqGetTimingConvertClockSourceSignal (TimingHandle *timingHandle*, char * *signal*, int * *signalLength*)

Get the name of the signal to use as conversion clock The signal name is device dependent.

Parameters:

timingHandle The handle to an existing timing object

signal The string to contain the current conversion clock signal

signalLength The length of the string, on function return contains the number of characters copied. If signal is NULL, signalLength contains the required length for signal

Returns:

the status code

See also:

UeiDaqSetTimingConvertClockSourceSignal (p. 945)

3.2.3.383 UeiDaqAPI int UeiDaqGetTimingConvertClockTimebaseDivisor (TimingHandle *timingHandle*, int * *divisor*)

Get the convert clock divisor. This divisor is used to divide an external clock using one of the on-board counter/timers.

Parameters:

timingHandle The handle to an existing timing object

divisor the timebase divisor

Returns:

the status code

See also:

UeiDaqSetTimingConvertClockTimebaseDivisor (p. 946)

3.2.3.384 UeiDaqAPI int UeiDaqGetTimingDuration (TimingHandle *timingHandle*, tUeiTimingDuration * *duration*)

Get the duration of the timed operation.

Parameters:

timingHandle The handle to an existing timing object

duration The returned duration

Returns:

The status code

See also:

UeiDaqGetTimingHandle (p. 839)

3.2.3.385 UeiDaqAPI int UeiDaqGetTimingHandle (SessionHandle *sessionHandle*, TimingHandle * *timing*)

Get a handle to the timing object associated with the session.

Parameters:

sessionHandle The handle to an existing session

timing A handle to the Timing object that is used to configure timing informations.

Returns:

The status code

3.2.3.386 UeiDaqAPI int UeiDaqGetTimingMode (TimingHandle *timingHandle*, tUeiTimingMode * *mode*)

Get the timing mode

Parameters:

timingHandle The handle to an existing timing object

mode The returned timing mode

Returns:

The status code

See also:

UeiDaqGetTimingHandle (p. 839)

3.2.3.387 UeiDaqAPI int UeiDaqGetTimingScanClockDestinationSignal (TimingHandle *timingHandle*, char * *signal*, int * *signalLength*)

Get the name of the signal driven by the Scan clock The signal name is device dependent.

Parameters:

timingHandle The handle to an existing timing object

signal The string to contain the current scan clock destination signal

signalLength The length of the string, on function return contains the number of characters copied. If signal is NULL, signalLength contains the required length for signal

Returns:

the status code

See also:

UeiDaqSetTimingScanClockDestinationSignal (p. 947)

3.2.3.388 UeiDaqAPI int UeiDaqGetTimingScanClockEdge (TimingHandle *timingHandle*, tUeiDigitalEdge * *edge*)

Get the active edge of the scan clock.

Parameters:

timingHandle The handle to an existing timing object

edge The returned active edge

Returns:

The status code

See also:

UeiDaqGetTimingHandle (p. 839)

3.2.3.389 UeiDaqAPI int UeiDaqGetTimingScanClockRate (TimingHandle *timingHandle*, f64 * *rate*)

Get the rate of the scan clock

Parameters:

timingHandle The handle to an existing timing object

rate The returned clock rate

Returns:

The status code

See also:

UeiDaqGetTimingHandle (p. 839)

3.2.3.390 UeiDaqAPI int UeiDaqGetTimingScanClockSource (TimingHandle *timingHandle*, tUeiTimingClockSource * *source*)

Get the source of the scan clock

Parameters:

timingHandle The handle to an existing timing object

source The returned clock source

Returns:

The status code

See also:

UeiDaqGetTimingHandle (p. 839)

3.2.3.391 UeiDaqAPI int UeiDaqGetTimingScanClockSourceSignal (TimingHandle *timingHandle*, char * *signal*, int * *signalLength*)

Get the name of the signal to use as Scan clock The signal name is device dependent.

Parameters:

timingHandle The handle to an existing timing object

signal The string to contain the current scan clock source signal

signalLength The length of the string, on function return contains the number of characters copied. If signal is NULL, signalLength contains the required length for signal

Returns:

the status code

See also:

UeiDaqSetTimingScanClockSourceSignal (p. 948)

3.2.3.392 UeiDaqAPI int UeiDaqGetTimingScanClockTimebaseDividingCounter (TimingHandle *timingHandle*, int * *counter*)

Get the scan clock dividing counter. This counter is used to divide an external clock signal.

Parameters:

timingHandle The handle to an existing timing object

counter the timebase dividing counter

Returns:

the status code

See also:

UeiDaqSetTimingScanClockTimebaseDividingCounter (p. 949)

3.2.3.393 UeiDaqAPI int UeiDaqGetTimingScanClockTimebaseDivisor (TimingHandle *timingHandle*, int * *divisor*)

Get the scan clock divisor. This divisor is used to divide an external clock using one of the on-board counter/timers.

Parameters:

timingHandle The handle to an existing timing object

divisor the timebase divisor

Returns:

the status code

See also:

UeiDaqSetTimingScanClockTimebaseDivisor (p. 949)

3.2.3.394 UeiDaqAPI int UeiDaqGetTimingTimeout (TimingHandle *timingHandle*, Int32 * *timeout*)

Get the delay after which the session is aborted.

Parameters:

timingHandle The handle to an existing timing object

timeout The returned timeout in ms

Returns:

The status code

See also:

UeiDaqGetTimingHandle (p. 839)

3.2.3.395 UeiDaqAPI int UeiDaqGetTriggerAction (TriggerHandle *triggerHandle*, tUeiTriggerAction * *pAction*)

Get the action to execute when the trigger will occur

Parameters:

triggerHandle The handle to an existing trigger object

pAction the current trigger action

Returns:

The status code

See also:

UeiDaqSetTriggerAction (p. 950) **UeiDaqGetStartTriggerHandle** (p. 833) **UeiDaqGetStopTriggerHandle** (p. 833)

3.2.3.396 UeiDaqAPI int UeiDaqGetTriggerChannel (TriggerHandle *triggerHandle*, Int32 * *pChannel*)

Get the channel to monitor for the trigger condition.

Parameters:

triggerHandle The handle to an existing trigger object

pChannel the current channel to monitor.

Returns:

The status code

See also:

UeiDaqSetTriggerChannel **UeiDaqGetStartTriggerHandle** (p. 833) **UeiDaqGetStopTriggerHandle** (p. 833)

3.2.3.397 UeiDaqAPI int UeiDaqGetTriggerCondition (TriggerHandle *triggerHandle*, tUeiTriggerCondition * *pCondition*)

Get the condition for the analog software trigger

Parameters:

triggerHandle The handle to an existing trigger object

pCondition the current condition

Returns:

The status code

See also:

UeiDaqSetTriggerCondition (p. 950) **UeiDaqGetStartTriggerHandle** (p. 833) **UeiDaqGetStopTriggerHandle** (p. 833)

3.2.3.398 UeiDaqAPI int UeiDaqGetTriggerDestinationSignal (TriggerHandle *triggerHandle*, char * *signal*, int * *signalLength*)

Get the signal where the trigger is routed out of the device. The signal name is device dependent.

Parameters:

triggerHandle The handle to an existing trigger object

signal The string to contain the current trigger destination signal

signalLength The length of the string, on function return contains the number of characters copied. If signal is NULL, signalLength contains the required length for signal

Returns:

The status code

See also:

UeiDaqSetTriggerDestinationSignal (p. 950)

3.2.3.399 UeiDaqAPI int UeiDaqGetTriggerDigitalEdge (TriggerHandle *triggerHandle*, tUeiDigitalEdge * *edge*)

Get the active edge of the trigger signal

Parameters:

triggerHandle The handle to an existing trigger object

edge The returned trigger edge

Returns:

The status code

See also:

UeiDaqGetStartTriggerHandle (p. 833) **UeiDaqGetStopTriggerHandle** (p. 833)

3.2.3.400 UeiDaqAPI int UeiDaqGetTriggerHysteresis (TriggerHandle *triggerHandle*, f64 * *pHysteresis*)

Get the hysteresis window scaled value. The hysteresis window is a noise filter. The trigger occurs when the signal leave the window.

Parameters:

triggerHandle The handle to an existing trigger object

pHysteresis the current hysteresis

Returns:

The status code

See also:

UeiDaqSetTriggerHysteresis (p. 951) **UeiDaqGetStartTriggerHandle** (p. 833) **UeiDaqGetStopTriggerHandle** (p. 833)

3.2.3.401 UeiDaqAPI int UeiDaqGetTriggerLevel (TriggerHandle *triggerHandle*, f64 * *pLevel*)

Get the scaled value at which the trigger occurs. The value is in the same unit as the measurement.

Parameters:

triggerHandle The handle to an existing trigger object

pLevel the current level

Returns:

The status code

See also:

UeiDaqSetTriggerLevel (p. 951) **UeiDaqGetStartTriggerHandle** (p. 833) **UeiDaqGetStopTriggerHandle** (p. 833)

3.2.3.402 UeiDaqAPI int UeiDaqGetTriggerSource (TriggerHandle *triggerHandle*, tUeiTriggerSource * *source*)

Get the source of the signal that will trigger the start of the session

Parameters:

triggerHandle The handle to an existing trigger object

source The returned trigger source

Returns:

The status code

See also:

UeiDaqGetStartTriggerHandle (p. 833) **UeiDaqGetStopTriggerHandle** (p. 833)

3.2.3.403 UeiDaqAPI int UeiDaqGetTriggerSourceSignal (TriggerHandle *triggerHandle*, char * *signal*, int * *signalLength*)

Get the signal used to transmit the trigger to the device. The signal name is device dependent.

Parameters:

triggerHandle The handle to an existing trigger object

signal The string to contain the current trigger source signal

signalLength The length of the string, on function return contains the number of characters copied. If signal is NULL, signalLength contains the required length for signal

Returns:

The status code

See also:

UeiDaqSetTriggerSourceSignal (p. 952)

3.2.3.404 UeiDaqAPI int UeiDaqGetVRADCMovingAverage (ChannelHandle *channelHandle*, int * *mvAvrg*)

Get the size of the moving average window applied to the VR sensor signal while it is measured.

Parameters:

channelHandle The handle to an existing VR channel

mvAvrg The current ADC moving average.

Returns:

The status code

3.2.3.405 UeiDaqAPI int UeiDaqGetVRADCRate (ChannelHandle *channelHandle*, double * *rate*)

Get the rate at which the VR sensor signal is measured.

Parameters:

channelHandle The handle to an existing VR channel

rate The current ADC rate.

Returns:

The status code

3.2.3.406 UeiDaqAPI int UeiDaqGetVRAPTMode (ChannelHandle *channelHandle*, tUeiVRAPTMode * *mode*)

APT finds the point in time where the VR sensor output voltage falls below a certain threshold. This point marks the beginning of the gap between two teeth.

Parameters:

channelHandle The handle to an existing VR channel

mode the current APT mode

Returns:

The status code

3.2.3.407 UeiDaqAPI int UeiDaqGetVRAPTThreshold (ChannelHandle *channelHandle*, double * *threshold*)

The APT threshold is used when APT mode is set to "Fixed"

Parameters:

channelHandle The handle to an existing VR channel

threshold The current APT threshold.

Returns:

The status code

3.2.3.408 UeiDaqAPI int UeiDaqGetVRAPTThresholdDivider (ChannelHandle *channelHandle*, int * *divider*)

The APT threshold divider is used when APT mode is set to "Logic"

Parameters:

channelHandle The handle to an existing VR channel

divider The current APT threshold divider.

Returns:

The status code

3.2.3.409 UeiDaqAPI int UeiDaqGetVRMode (ChannelHandle *channelHandle*, tUeiVRMode * *mode*)

The VR-608 can use four different modes to measure velocity, position or direction The counting mode can be set to: Decoder: Even and Odd channels are used in pair to determine direction and position Timed: Count number of teeth detected during a timed interval N pulses: Measure the time taken to detect N teeth (Number of teeth needs to be set separately) Z pulse: Measure the number of teeth and the time elapsed between two Z pulses (The Z tooth is usually a gap or a double tooth on the encoder wheel)

In decoder mode the settings for the even channel are applied to both even and odd channels.

Parameters:

channelHandle The handle to an existing VR channel

mode the current VR mode

Returns:

The status code

3.2.3.410 UeiDaqAPI int UeiDaqGetVRNumberOfTeeth (ChannelHandle *channelHandle*, int * *numTeeth*)

The number of teeth on the encoder wheel

Parameters:

channelHandle The handle to an existing VR channel

numTeeth The current number of teeth.

Returns:

The status code

3.2.3.411 UeiDaqAPI int UeiDaqGetVRZCLevel (ChannelHandle *channelHandle*, double * *level*)

The ZC level is used when ZC mode is set to "Fixed"

Parameters:

channelHandle The handle to an existing VR channel

level The current ZC level.

Returns:

The status code

3.2.3.412 UeiDaqAPI int UeiDaqGetVRZCMode (ChannelHandle *channelHandle*, tUeiVRZCMode * *mode*)

Zero crossing finds the point in time where the VR sensor output voltage goes from positive to negative voltage. This point is when the center of the tooth is lining up with the center of the VR sensor.

Parameters:

channelHandle The handle to an existing VR channel

mode the current Zero Crossing mode

Returns:

The status code

3.2.3.413 UeiDaqAPI int UeiDaqIsAccelLowPassFilterEnabled (ChannelHandle *channelHandle*, int * *pEnabled*)

Get the low-pass filter state

Parameters:

channelHandle The handle to an existing accelerometer channel

pEnabled The low-pass filter state

Returns:

The status code.

See also:

UeiDaqEnableAccelLowPassfilter (p. 736)

3.2.3.414 UeiDaqAPI int UeiDaqIsAIChannelAutoZeroEnabled (ChannelHandle *channelHandle*, int * *enabled*)

Some analog input devices come with a special channel to measure ground offset. auto-zero substract the ground voltage measurement from the input voltage measurement.

Parameters:

channelHandle The handle to an existing channel

enabled the auto-zero state

Returns:

The status code

3.2.3.415 UeiDaqAPI int UeiDaqIsAIChannelBiasEnabled (ChannelHandle *channelHandle*, int * *enabled*)

Bias resistors connects the negative input to ground and positive input to a voltage source. This is useful to measure temperature from un-grounded thremocouple. Disable bias to measure from grounded thermocouples

Parameters:

channelHandle The handle to an existing channel

enabled true if bias is enabled, false otherwise

Returns:

The status code

3.2.3.416 UeiDaqAPI int UeiDaqIsAIChannelCircuitOpen (ChannelHandle *channelHandle*, int * *open*)

Get result of open circuit detection

Parameters:

channelHandle The handle to an existing channel

open true if circuit is open, false otherwise

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.417 UeiDaqAPI int UeiDaqIsAIChannelOpenCircuitTestEnabled (ChannelHandle *channelHandle*, int * *enabled*)

Get status of open circuit detection

Parameters:

channelHandle The handle to an existing channel

enabled true if open circuit detection is enabled, false otherwise

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.418 UeiDaqAPI int UeiDaqIsAIVExOffsetNullingEnabled (ChannelHandle *channelHandle*, int * *nullingEnabled*)

Determines whether the offset nulling circuitry is enabled. With Offset nulling enabled, a nulling circuit adds an adjustable DC voltage to the output of the amplifier making sure that the bridge output is 0V when no strain is applied.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

nullingEnabled 1 if nulling is enabled, 0 otherwise.

Returns:

The status code.

See also:

UeiDaqEnableAIVExOffsetNulling (p. 738)

3.2.3.419 UeiDaqAPI int UeiDaqIsAIVExShuntCalibrationEnabled (ChannelHandle *channelHandle*, int * *pIsShuntCalEnabled*)

Determines whether the shunt calibration resistor is connected.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

pIsShuntCalEnabled true the shunt resistor is enabled, false otherwise

Returns:

The status code.

See also:

UeiDaqEnableAIVExShuntCalibration (p. 738)

3.2.3.420 UeiDaqAPI int UeiDaqIsAOChannelDefaultValueEnabled (ChannelHandle *channelHandle*, int * *pDefaultValueEnabled*)

Determines whether output will be set to a default value when session stops.

Parameters:

channelHandle The handle to an existing channel

pDefaultValueEnabled The default value state.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.421 UeiDaqAPI int UeiDaqIsAOProtectedCircuitBreakerEnabled (ChannelHandle *channelHandle*, int *index*, int * *pEnabled*)

Return true is circuit breaker for this channels is enabled and false otherwise

Parameters:

channelHandle The handle to an existing channel

index The 0 based index of the circuit breaker

pEnabled The current circuit breaker state

Returns:

The status code

3.2.3.422 UeiDaqAPI int UeiDaqIsARINCInputLabelFilterEnabled (ChannelHandle *channelHandle*, int * *pLabelFilterEnabled*)

Determines whether label filtering is enabled.

Parameters:

channelHandle The handle to an existing ARINC-429 input channel

pLabelFilterEnabled The label filter state.

Returns:

The status code.

See also:

UeiDaqEnableARINCInputLabelFilter (p. 739)

3.2.3.423 UeiDaqAPI int UeiDaqIsARINCInputSDIFilterEnabled (ChannelHandle *channelHandle*, int * *pSDIFilterEnabled*)

Determines whether the SDI filter is turned on or off.

Parameters:

channelHandle The handle to an existing ARINC-429 input channel

pSDIFilterEnabled true if SDI filtering is on, false otherwise

Returns:

The status code.

See also:

UeiDaqEnableARINCInputSDIFilter (p. 739)

3.2.3.424 UeiDaqAPI int UeiDaqIsARINCInputSlowSlewRateEnabled (ChannelHandle *channelHandle*, int * *pSlowSlewRateEnabled*)

Determines whether slow slew rate is enabled.

Parameters:

channelHandle The handle to an existing ARINC-429 input channel

pSlowSlewRateEnabled 1 if slow slew rate is on, 0 otherwise

Returns:

The status code.

See also:

UeiDaqEnableARINCInputSlowSlewRate (p. 740)

3.2.3.425 UeiDaqAPI int UeiDaqIsARINCInputTimestampingEnabled (ChannelHandle *channelHandle*, int * *pTimestampingEnabled*)

Determines whether each received frame is timestamped.

Parameters:

channelHandle The handle to an existing ARINC-429 input channel

pTimestampingEnabled 1 if timestamping is on, 0 otherwise

Returns:

The status code.

See also:

UeiDaqEnableARINCInputTimestamping (p. 740)

3.2.3.426 UeiDaqAPI int UeiDaqIsARINCOutputLoopbackEnabled (ChannelHandle *channelHandle*, int * *pLoopbackEnabled*)

Determines whether loopback is enabled. When enabled you can read back packet sent to this port on a dedicated loopback port.

Parameters:

channelHandle The handle to an existing ARINC-429 output channel

pLoopbackEnabled 1 if loopback is enabled, 0 if it is disabled.

Returns:

The status code.

See also:

UeiDaqEnableARINCOutputLoopback (p. 740)

3.2.3.427 UeiDaqAPI int UeiDaqIsARINCOutputSchedulerEnabled (ChannelHandle *channelHandle*, int * *pSchedulerEnabled*)

Determines whether scheduling is enabled.

Parameters:

channelHandle The handle to an existing ARINC-429 output channel

pSchedulerEnabled The scheduler state.

Returns:

The status code.

See also:

UeiDaqEnableARINCOutputScheduler (p. 741)

3.2.3.428 UeiDaqAPI int UeiDaqIsARINCOutputSlowSlewRateEnabled (ChannelHandle *channelHandle*, int * *pSlowSlewRateEnabled*)

Determines whether slow slew rate is enabled.

Parameters:

channelHandle The handle to an existing ARINC-429 output channel

pSlowSlewRateEnabled 1 if slow slew rate is on, 0 otherwise

Returns:

The status code.

See also:

UeiDaqEnableARINCOutputSlowSlewRate (p. 741)

3.2.3.429 UeiDaqAPI int UeiDaqIsCANPortWarningAndErrorLoggingEnabled (ChannelHandle *channelHandle*, int * *pEnabled*)

When logging is enabled, bus warnings and errors are sent in the data stream. Use the Type field of the CANDataFrame structure to determine whether received frames are data or error frames.

Parameters:

channelHandle The handle to an existing CAN port

pEnabled 1 if logging is enabled, 0 otherwise.

See also:

UeiDaqEnableCANPortWarningAndErrorLogging (p. 742)

3.2.3.430 UeiDaqAPI int UeiDaqIsDataStreamBlockingEnabled (DataStreamHandle *dataStreamHandle*, Int32 * *enabled*)

When enabled, read operations will block until the requested amount of data is received from the device. when disabled, read operations return immediately with the amount of data available.

Parameters:

dataStreamHandle The handle to an existing data stream

enabled the blocking setting value.

Returns:

The status code

3.2.3.431 UeiDaqAPI int UeiDaqIsDIIndustrialACModeEnabled (ChannelHandle *channelHandle*, int *line*, int * *pAcMode*)

Verify whether AC mode is enabled or disabled In AC mode, the RMS voltage measured at each input is compared with lowa nd high thresholds to determine the state of the input

Parameters:

channelHandle The handle to an existing industrial DI channel

line The input line to query

pAcMode the AC mode status

Returns:

The status code.

3.2.3.432 UeiDaqAPI int UeiDaqIsDMMProtectionReconfiguredSafeRange (SessionHandle *sessionHandle*, int * *isReconfigured*)

Read DMM status returned with last read data and return if protection reconfigured to safe range when data was read.

Parameters:

sessionHandle The handle to an existing session

isReconfigured protection reconfigured to safe range status return The status code

3.2.3.433 UeiDaqAPI int UeiDaqIsDMMProtectionTripped (SessionHandle *sessionHandle*, int * *isTripped*)

Read DMM status returned with last read data and return if protection was tripped when data was read.

Parameters:

sessionHandle The handle to an existing session

isTripped protection tripped status return The status code

3.2.3.434 UeiDaqAPI int UeiDaqIsDMMProtectionTrippedMaxRange (SessionHandle *sessionHandle*, int * *isTrippedMax*)

Read DMM status returned with last read data and return if protection was tripped in max range when data was read.

Parameters:

sessionHandle The handle to an existing session

isTrippedMax protection tripped in max range status return The status code

3.2.3.435 UeiDaqAPI int UeiDaqIsDOChannelDefaultValueEnabled (ChannelHandle *channelHandle*, int * *pDefaultValueEnabled*)

Determines whether output will be set to a default value when session stops.

Parameters:

channelHandle The handle to an existing channel

pDefaultValueEnabled The default value state.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.436 UeiDaqAPI int UeiDaqIsDOProtectedCircuitBreakerEnabled (ChannelHandle *channelHandle*, int *line*, int * *pEnabled*)

Return true is circuit breaker for this output line is enabled and false otherwise

Parameters:

channelHandle The handle to an existing channel

line The output line to configure within the port

pEnabled The current circuit breaker state

Returns:

The status code

3.2.3.437 UeiDaqAPI int UeiDaqIsI2CMasterSecureShellEnabled (ChannelHandle *channelHandle*, int * *enabled*)

Get the I2C secure shell setting

Parameters:

channelHandle The handle to an I2C master port

enabled The current I2C secure shell setting

Returns:

The status code

3.2.3.438 UeiDaqAPI int UeiDaqIsI2CMasterTerminationResistorEnabled (ChannelHandle *channelHandle*, int * *enabled*)

Get the I2C termination resistor setting

Parameters:

channelHandle The handle to an I2C master port

resistance The current I2C termination resistor setting

Returns:

The status code

3.2.3.439 UeiDaqAPI int UeiDaqIsI2CMultiMasterEnabled (ChannelHandle *channelHandle*, int * *enabled*)

Get the I2C multi-master setting

Parameters:

channelHandle The handle to an I2C master port

enabled The current I2C multi-master setting

Returns:

The status code

3.2.3.440 UeiDaqAPI int UeiDaqIsI2CSlaveAddressClockStretchingEnabled (ChannelHandle *channelHandle*, int * *enabled*)

Query the status of clock stretching during address cycle.

Parameters:

channelHandle The handle to an I2C master port
enabled the status of clock stretching during address cycle.

Returns:

The status code

3.2.3.441 UeiDaqAPI int UeiDaqIsI2CSlaveBusMonitorAckEnabled (ChannelHandle *channelHandle*, int * *enabled*)

Get the I2C bus monitor ACK setting

Parameters:

channelHandle The handle to an I2C slave port
enabled The current I2C bus monitor ACK setting

Returns:

The status code

3.2.3.442 UeiDaqAPI int UeiDaqIsI2CSlaveBusMonitorEnabled (ChannelHandle *channelHandle*, int * *enabled*)

Get the I2C bus monitor setting

Parameters:

channelHandle The handle to an I2C slave port
enabled The current I2C bus monitor setting

Returns:

The status code

3.2.3.443 UeiDaqAPI int UeiDaqIsI2CSlaveReceiveClockStretchingEnabled (ChannelHandle *channelHandle*, int * *enabled*)

Query the status of clock stretching during receive cycle.

Parameters:

channelHandle The handle to an I2C master port
enabled the status of clock stretching during receive cycle.

Returns:

The status code

3.2.3.444 UeiDaqAPI int UeiDaqIsI2CSlaveTransmitClockStretchingEnabled (ChannelHandle *channelHandle*, int * *enabled*)

Query the status of clock stretching during transmit cycle.

Parameters:

channelHandle The handle to an I2C master port

enabled the status of clock stretching during transmit cycle.

Returns:

The status code

3.2.3.445 UeiDaqAPI int UeiDaqIsIRIGTimeKeeperAutoFollowEnabled (ChannelHandle *channelHandle*, int * *autoFollowEnabled*)

If selected external 1PPS source does not deliver pulses (because of a break in timecode transmission, for example). Timekeeper can switch to internal timebase when externally derived one is not available

Parameters:

channelHandle The handle to an existing MIL-1553 channel

autoFollowEnabled The auto-follow state.

Returns:

The status code

See also:

UeiDaqEnableIRIGAutoFollow

3.2.3.446 UeiDaqAPI int UeiDaqIsIRIGTimeKeeperInvalidDayEnabled (ChannelHandle *channelHandle*, int * *invalidDayEnabled*)

Select whether days information is invalid in the input timecode

Parameters:

channelHandle The handle to an existing IRIG channel

invalidDayEnabled The invalid day state.

Returns:

The status code

See also:

UeiDaqEnableIRIGInvalidDay

3.2.3.447 UeiDaqAPI int UeiDaqIsIRIGTimeKeeperInvalidMinuteEnabled (ChannelHandle *channelHandle*, int * *invalidMinuteEnabled*)

Select whether minutes information is invalid in the input timecode

Parameters:

channelHandle The handle to an existing IRIG channel

invalidMinuteEnabled The invalid minute state.

Returns:

The status code

See also:

UeiDaqEnableIRIGInvalidMinute

3.2.3.448 UeiDaqAPI int UeiDaqIsIRIGTimeKeeperInvalidSecondEnabled (ChannelHandle *channelHandle*, int * *invalidSecondEnabled*)

Select whether seconds information is invalid in the input timecode

Parameters:

channelHandle The handle to an existing IRIG channel

invalidSecondEnabled The invalid second state.

Returns:

The status code

See also:

UeiDaqEnableIRIGInvalidSecond

3.2.3.449 UeiDaqAPI int UeiDaqIsIRIGTimeKeeperNominalValueEnabled (ChannelHandle *channelHandle*, int * *nominalValueEnabled*)

Select whether to use nominal period (i.e. 100e6 pulses of 100MHz base clock) or the period measured by timekeeper (it measures and averages number of base clock cycles between externally derived 1PPS pulses when they are valid).

Parameters:

channelHandle The handle to an existing MIL-1553 channel

nominalValueEnabled The nominal value state.

Returns:

The status code

See also:

UeiDaqEnableIRIGNominalValue

3.2.3.450 UeiDaqAPI int UeiDaqIsIRIGTimeKeeperSBSEnabled (ChannelHandle *channelHandle*, int * *SBSenabled*)

Select whether to use SBS timecode section for TimeKeeper hour/min decoding (if BCD data in the incoming timecode is corrupted)

Parameters:

channelHandle The handle to an existing IRIG input channel

SBSenabled The SBS state.

Returns:

The status code

See also:

UeiDaqEnableIRIGSBS

3.2.3.451 UeiDaqAPI int UeiDaqIsIRIGTimeKeeperSubPPSEnabled (ChannelHandle *channelHandle*, int * *subPPSEnabled*)

Select whether external timebase is slower than 1PPS or is not derived from the timecode

Parameters:

channelHandle The handle to an existing MIL-1553 channel

subPPSEnabled The sub PPS state.

Returns:

The status code

See also:

UeiDaqEnableIRIGSubPPS

3.2.3.452 UeiDaqAPI int UeiDaqIsLVDTExternalExcitationEnabled (ChannelHandle *channelHandle*, int * *pEnabled*)

Determines whether the excitation will be provided by the acquisition device or by an external source

Parameters:

channelHandle The handle to an existing LVDT channel

pEnabled The current external excitation state

Returns:

The status code.

See also:

UeiDaqEnableLVDTExternalExcitation (p. 748)

3.2.3.453 UeiDaqAPI int UeiDaqIsMIL1553FilterEnabled (ChannelHandle *channelHandle*, int * *pFilterEnabled*)

Determines whether frame filtering is enabled.

Parameters:

channelHandle The handle to an existing MIL-1553 input channel

pFilterEnabled The label filter state.

Returns:

The status code.

See also:

UeiDaqEnableMIL1553Filter (p. 748)

3.2.3.454 UeiDaqAPI int UeiDaqIsMIL1553SchedulerEnabled (ChannelHandle *channelHandle*, int * *pSchedulerEnabled*)

Determines whether scheduling is enabled.

Parameters:

channelHandle The handle to an existing MIL-1553 channel

pSchedulerEnabled The scheduler state.

Returns:

The status code.

See also:

UeiDaqEnableMIL1553Scheduler (p. 749)

3.2.3.455 UeiDaqAPI int UeiDaqIsMIL1553TimestampingEnabled (ChannelHandle *channelHandle*, int * *pTimestampingEnabled*)

Determines whether each received frame is timestamped.

Parameters:

channelHandle The handle to an existing MIL-1553 channel

pTimestampingEnabled 1 if timestamping is on, 0 otherwise

Returns:

The status code.

See also:

UeiDaqEnableMIL1553Timestamping (p. 749)

3.2.3.456 UeiDaqAPI int UeiDaqIsMuxBreakBeforeMakeEnabled (ChannelHandle *channelHandle*, int * *enabled*)

Get the current port break before make setting

Parameters:

channelHandle The handle to a MUX port

enabled the current break before make

Returns:

The status code

3.2.3.457 UeiDaqAPI int UeiDaqIsMuxSyncInputEdgeModeEnabled (ChannelHandle *channelHandle*, int * *enabled*)

Get the current port synchronization input edge/level setting The sync input can work in level or edge mode

Parameters:

channelHandle The handle to a MUX port

enabled the current synchronization input

Returns:

The status code

3.2.3.458 UeiDaqAPI int UeiDaqIsMuxSyncInputEnabled (ChannelHandle *channelHandle*, int * *enabled*)

Get the current port synchronization input setting A MUX device can wait for a pulse on its synchronization input before configuring its relays

Parameters:

channelHandle The handle to a MUX port

enabled the current synchronization input

Returns:

The status code

3.2.3.459 UeiDaqAPI int UeiDaqIsSerialPortErrorReportingEnabled (ChannelHandle *channelHandle*, int * *pEnabled*)

Determines whether the framing errors or parity errors will be reported

Parameters:

channelHandle The handle to an existing serial port

pEnabled The error reporting state.

Returns:

The status code.

See also:

UeiDaqEnableSerialPortErrorReporting (p. 750)

3.2.3.460 UeiDaqAPI int UeiDaqIsSerialPortHDEchoSuppressionEnabled (ChannelHandle *channelHandle*, int * *pEnabled*)

Determines whether the half duplex echo suppression is enabled

Parameters:

channelHandle The handle to an existing serial port

pEnabled The half-duplex echo suppression state.

Returns:

The status code.

See also:

UeiDaqEnableSerialPortHDEchoSuppression (p. 751)

3.2.3.461 UeiDaqAPI int UeiDaqIsSerialPortOnTheFlyParityBitEnabled (ChannelHandle *channelHandle*, int * *pEnabled*)

Determines whether the parity bit can be specified on the fly as part of the data stream (using uInt16 data instead of uInt8)

Parameters:

channelHandle The handle to an existing serial port

pEnabled The parity bit state.

Returns:

The status code.

See also:

UeiDaqEnableSerialPortOnTheFlyParityBit (p. 751)

3.2.3.462 UeiDaqAPI int UeiDaqIsSerialPortRxTerminationResistorEnabled (ChannelHandle *channelHandle*, int * *pEnabled*)

Determines whether the RX termination resistor is enabled

Parameters:

channelHandle The handle to an existing serial port

pEnabled The RX termination resistor state.

Returns:

The status code.

See also:

UeiDaqEnableSerialPortRxTerminationResistor (p. 751)

3.2.3.463 UeiDaqAPI int UeiDaqIsSerialPortTxAutoDisableEnabled (ChannelHandle *channelHandle*, int * *pEnabled*)

Determines whether transmitter should automatically disable itself whenever the TX FIFO is empty (only used in RS-485 full duplex mode) This allows multiple RS-485 devices to share the same bus

Parameters:

channelHandle The handle to an existing serial port

pEnabled The TX auto-disable state

Returns:

The status code.

See also:

UeiDaqEnableSerialPortTxAutoDisable (p. 752)

3.2.3.464 UeiDaqAPI int UeiDaqIsSerialPortTxTerminationResistorEnabled (ChannelHandle *channelHandle*, int * *pEnabled*)

Determines whether the TX termination resistor is enabled

Parameters:

channelHandle The handle to an existing serial port

pEnabled The TX termination resistor state.

Returns:

The status code.

See also:

UeiDaqEnableSerialPortTxTerminationResistor (p. 752)

3.2.3.465 UeiDaqAPI int UeiDaqIsSessionRunning (SessionHandle *sessionHandle*, int * *done*)

Check whether the session is in the running state.

Parameters:

sessionHandle The handle to an existing session

done true if the session is still running and false otherwise

Returns:

The status code

See also:

UeiDaqStartSession (p. 955) **UeiDaqStopSession** (p. 955) UeiDaqCleanUp **UeiDaqWaitUntilSessionIsDone** (p. 957)

3.2.3.466 UeiDaqAPI int UeiDaqIsSimulatedSynchroResolverExternalExcitationEnabled (ChannelHandle *channelHandle*, int * *pEnabled*)

Determines whether the excitation will be provided by the acquisition device or by an external source

Parameters:

channelHandle The handle to an existing simulated synchro/resolver channel

pEnabled the external excitation state.

Returns:

The status code.

See also:

UeiDaqEnableSimulatedSynchroResolverExternalExcitation (p. 753)

3.2.3.467 UeiDaqAPI int UeiDaqIsSynchroResolverExternalExcitationEnabled (ChannelHandle *channelHandle*, int * *pEnabled*)

Determines whether the excitation will be provided by the acquisition device or by an external source

Parameters:

channelHandle The handle to an existing synchro/resolver channel

pEnabled the external excitation state.

Returns:

The status code.

See also:

UeiDaqEnableSynchroResolverExternalExcitation (p. 753)

3.2.3.468 UeiDaqAPI int UeiDaqLoadSessionFromFile (SessionHandle *sessionHandle*, char * *sessionFile*)

Load Session settings from a file, session is ready to be started

Parameters:

sessionHandle The handle of the session

sessionFile String containing the session settings file path

Returns:

The status code

3.2.3.469 UeiDaqAPI int UeiDaqLoadSessionFromXml (SessionHandle *sessionHandle*, char * *xmlSettings*)

Load Session settings from an XML stream, session is ready to be started

Parameters:

sessionHandle The handle of the session

xmlSettings String containing the session settings in XML format

Returns:

The status code

3.2.3.470 UeiDaqAPI int UeiDaqParseResource (char * *resource*, char * *deviceClass*, char * *remoteAddress*, int * *deviceID*, tUeiSessionType * *type*, int * *channelList*, int * *channelListSize*)

Parse the resource string and return its components. You can pass NULL for the components you are not interested in. When the function is called channelListSize should contain the size of the channelList array. When the function returns channelListSize contains the actual number of elements copied to channelList

Parameters:

resource The resource string

deviceClass The name of the device class

remoteAddress The remote address, NULL if the ersource is local

deviceID The ID number of the device

type The session type

channelList An array of integer representing the channel list

channelListSize The size of the channel list

Returns:

The status code

See also:

UeiDaqEnumDriver (p. 754)

3.2.3.471 UeiDaqAPI int UeiDaqPauseSession (SessionHandle *sessionHandle*, int *port*, uInt32 * *numValuesPending*)

Pause the specified messaging port.

Parameters:

sessionHandle The handle to an existing session

port The port to pause

numValuesPending The number of values pending at the time of the pause. On some device, those values will need to be resent

Returns:

The status code

3.2.3.472 UeiDaqAPI int UeiDaqReadARINCRawWords (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numWords*, uInt32 * *pBuffer*, Int32 * *numWordsRead*)

Read ARINC raw words from the stream. Raw ARINC word is a 32 bits value encoding the ARINC word fields: SSM, SDI, Label, Data and Parity It is the caller's responsability to allocate enough memory in the buffer. to store the words

Parameters:

sessionHandle The handle to an existing session

port the port to read the message from

numWords Number of words to read

pBuffer Destination buffer for the words

numWordsRead Number of words actually read from the port

Returns:

The status code

3.2.3.473 UeiDaqAPI int UeiDaqReadARINCWords (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numWords*, tUeiARINCWord * *pBuffer*, Int32 * *numWordsRead*)

Read ARINC words from the stream. It is the caller's responsability to allocate enough memory in the buffer. to store the words

Parameters:

sessionHandle The handle to an existing session

port the port to read the message from

numWords Number of words to read

pBuffer Destination buffer for the words

numWordsRead Number of words actually read from the port

Returns:

The status code

3.2.3.474 UeiDaqAPI int UeiDaqReadARINCWordsAsync (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numWords*, tUeiARINCWord * *pBuffer*, tUeiEventCallback *pEventCallback*)

Read ARINC words asynchronously from the stream. This function returns immediately, the specified callback function is called once the buffer is ready. The callback function parameter contains the number of words actually read. It is the caller's responsability to allocate enough memory in the buffer. to store the words

Parameters:

sessionHandle The handle to an existing session

port the port to read the message from

numWords Number of words to read

pBuffer Destination buffer for the words

pEventCallback The callback function pointer

Returns:

The status code

3.2.3.475 UeiDaqAPI int UeiDaqReadCANFrame (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numFrames*, tUeiCANFrame * *pBuffer*, Int32 * *numFramesRead*)

Read CAN frames from the stream. It is the caller's responsability to allocate enough memory in the buffer. to store the frames

Parameters:

sessionHandle The handle to an existing session

port the port to read the message from

numFrames Number of frames to read

pBuffer Destination buffer for the frames

numFramesRead Number of frames actually read from the port

Returns:

The status code

3.2.3.476 UeiDaqAPI int UeiDaqReadCANFrameAsync (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numFrames*, tUeiCANFrame * *pBuffer*, tUeiEventCallback *pEventCallback*)

Read CAN frames asynchronously from the stream. This function returns immediately, the specified callback function is called once the buffer is ready. The callback function parameter contains the number of frames actually read. It is the caller's responsability to allocate enough memory in the buffer. to store the frames

Parameters:

sessionHandle The handle to an existing session

port the port to read the message from
numFrames Number of frames to read
pBuffer Destination buffer for the frames
pEventCallback The callback function pointer

Returns:

The status code

3.2.3.477 UeiDaqAPI int UeiDaqReadCSDBFrame (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numMessages*, tUeiCSDBMessage * *pBuffer*, Int32 * *numMessagesRead*)

Read CSDB message blocks from the stream. It is the caller's responsability to allocate enough memory in the buffer. to store the messages

Parameters:

sessionHandle The handle to an existing session
port the port to read the message from
numMessages Number of message blocks to read
pBuffer Destination buffer for the messages
numMessagesRead Number of message blocks actually read from the port

Returns:

The status code

3.2.3.478 UeiDaqAPI int UeiDaqReadDeviceEEPROM (DeviceHandle *deviceHandle*, int *bank*, int *subSystem*, uInt8 * *buffer*, int * *size*, tUeiEEPROMArea *area*)

Read the device EEPROM content. EEPROM contains informations such as Manufacture date, calibration date and calibration data The EEPROM content is returned as an opaque array of bytes. Each device has a unique way of storing informations in its EEPROM and it is up to the calling application to interpret its content.

Some devices have multiple EEPROM areas. Use the bank parameter to specify which EEPROM you wish to read from

Parameters:

deviceHandle The handle to an existing device
bank The EEPROM bank to read from
subSystem The sub-system to read from
buffer Buffer big enough to store the EEPROM content
size The number of bytes read
area The area to access in the EEPROM

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.479 UeiDaqAPI int UeiDaqReadDeviceRAM (DeviceHandle *deviceHandle*, uInt32 *address*, int *size*, uInt8 * *buffer*)

Read from the device RAM. Only for devices that actually have RAM.

Parameters:

deviceHandle The handle to an existing device

address address of the first element to read

buffer Buffer to store the RAM content

size The number of bytes to read from RAM

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.480 UeiDaqAPI int UeiDaqReadDeviceRegister32 (DeviceHandle *deviceHandle*, uInt32 *offset*, uInt32 * *value*)

Read a device register as a 32 bits value.

Parameters:

deviceHandle The handle to an existing device

offset Offset of the register relative to device's base address

value The register content

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.481 UeiDaqAPI int UeiDaqReadDMMStatus (SessionHandle *sessionHandle*, uInt32 * *pStatus*)

Read DMM status returned with last read data. Status is only updated when channel data is read.

Parameters:

sessionHandle The handle to an existing session

pStatus destination for status

Returns:

The status code

3.2.3.482 UeiDaqAPI int UeiDaqReadHDLCFrame (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numBytes*, void * *pBuffer*, Int32 * *numBytesRead*)

Read HDLC frame from the stream. It is the caller's responsability to allocate enough memory in the buffer. to store the message

Parameters:

sessionHandle The handle to an existing session
port the port to read the message from
numBytes Number of bytes to read
pBuffer Destination buffer for the messages
numBytesRead Number of bytes actually read from the port

Returns:

The status code

3.2.3.483 UeiDaqAPI int UeiDaqReadI2CMaster (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numMessages*, tUeiI2CMasterMessage * *pBuffer*, Int32 * *numMessagesRead*)

Read messages received by master.

Parameters:

sessionHandle The handle to an existing session
port The I2C master port to read from
numMessages The number of messages to read
pBuffer Destination buffer
numMessagesRead The number of messages recevied

Returns:

The status code

3.2.3.484 UeiDaqAPI int UeiDaqReadI2CSlave (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numMessages*, tUeiI2CSlaveMessage * *pBuffer*, Int32 * *numMessagesRead*)

Read messages received by slave.

Parameters:

sessionHandle The handle to an existing session
port The I2C slave port to read from
numMessages The number of messages to read
pBuffer Destination buffer
numMessagesRead The number of messages received

Returns:

The status code

3.2.3.485 UeiDaqAPI int UeiDaqReadIRIGGPS (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numBytes*, char * *pBuffer*, Int32 * *numBytesRead*)

Read NMEA data from GPS associated with specified port

Parameters:

sessionHandle The handle to an existing session

port The port to read from

numBytes maximum number of bytes to read

pBuffer destination buffer

numBytesRead Number of bytes read from GPS receiver

Returns:

The status code

3.2.3.486 UeiDaqAPI int UeiDaqReadIRIGRawTimeCode (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numTimeCodeChars*, uInt32 * *pBuffer*, Int32 * *numTimeCodeCharsRead*)

Read raw input time code (for internal use only)

Parameters:

sessionHandle The handle to an existing session

port The port to read from

numTimeCodeChars number of time code characters to read

pBuffer destination buffer

numTimeCodeCharsRead Number of time code characters actually read

Returns:

The status code

3.2.3.487 UeiDaqAPI int UeiDaqReadIRIGTimeKeeperBCDTime (SessionHandle *sessionHandle*, Int32 *port*, tUeiBCDTime * *pBuffer*, Int32 * *status*)

Read current time in Binary Coded Decimal format from IRIG layer

Parameters:

sessionHandle The handle to an existing session

port The port to read from

pBuffer Destination buffer for the time

status The time keeper status

Returns:

The status code

3.2.3.488 UeiDaqAPI int UeiDaqReadIRIGTimeKeeperCANSITime (SessionHandle *sessionHandle*, Int32 *port*, tUeiANSITime * *pBuffer*, Int32 * *status*)

Read current time in C-ANSI format from IRIG layer

Parameters:

sessionHandle The handle to an existing session

port The port to read from

pBuffer Destination buffer for the time

status The time keeper status

Returns:

The status code

3.2.3.489 UeiDaqAPI int UeiDaqReadIRIGTimeKeeperSBSTime (SessionHandle *sessionHandle*, Int32 *port*, tUeiSBSTime * *pBuffer*, Int32 * *status*)

Read current time in Straight Binary Seconds format from IRIG layer

Parameters:

sessionHandle The handle to an existing session

port The port to read from

pBuffer Destination buffer for the time

status The time keeper status

Returns:

The status code

3.2.3.490 UeiDaqAPI int UeiDaqReadIRIGTReg (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numRegisters*, uInt32 * *pBuffer*, Int32 * *numRegistersRead*)

For internal use only

Parameters:

sessionHandle The handle to an existing session

port The port to read from

numRegisters number of registers to read

pBuffer destination buffer

numRegistersRead Number of registers actually read

Returns:

The status code

3.2.3.491 UeiDaqAPI int UeiDaqReadLVDTCoilAmplitudes (SessionHandle *sessionHandle*, f64 * *pPrimaryCoilsBuffer*, f64 * *pSecondaryCoilsBuffer*)

Read RMS amplitudes for the primary and secondary coils of each configured channel

Parameters:

sessionHandle The handle to an existing session
channel The channel to read from
pPrimaryCoilsBuffer destination buffer for primary coils amplitudes
pSecondaryCoilsBuffer destination buffer for secondary coils amplitudes

Returns:

The status code

3.2.3.492 UeiDaqAPI int UeiDaqReadMessage (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numBytes*, void * *pBuffer*, Int32 * *numBytesRead*)

Read message from the stream. It is the caller's responsability to allocate enough memory in the buffer. to store the message

Parameters:

sessionHandle The handle to an existing session
port the port to read the message from
numBytes Number of bytes to read
pBuffer Destination buffer for the messages
numBytesRead Number of bytes actually read from the port

Returns:

The status code

3.2.3.493 UeiDaqAPI int UeiDaqReadMessageAsync (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numBytes*, void * *pBuffer*, tUeiEventCallback *pEventCallback*)

Read messages asynchronously from the stream. This function returns immediately, the specified callback function is called once the buffer is ready. The callback function parameter contains the number of bytes actually read. It is the caller's responsability to allocate enough memory in the buffer. to store the message

Parameters:

sessionHandle The handle to an existing session
port the port to read the message from
numBytes Number of bytes to read
pBuffer Destination buffer for the messages
pEventCallback The callback function pointer

Returns:

The status code

3.2.3.494 UeiDaqAPI int UeiDaqReadMIL1553A708DataFrames (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numFrames*, tUeiMIL1553A708DataFrame * *pBuffer*, Int32 * *numFramesRead*)

Read ARINC 708 data frames from the stream. The device assiociated with this data stream must support ARINC 708 It is the caller's responsability to allocate enough memory in the buffer. to store the frames

Parameters:

sessionHandle The handle to an existing session

port The port to read from

numFrames Maximum number of frames that can be stored in the buffer

pBuffer Destination buffer for the frames

numFramesRead The actual number of frames read

Returns:

The status code

3.2.3.495 UeiDaqAPI int UeiDaqReadMIL1553BCCBStatusFrames (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numFrames*, tUeiMIL1553BCCBStatusFrame * *pBuffer*, Int32 * *numFramesRead*)

Read MIL-1553 RT status from the stream. The device associated with this data stream must support MIL-1553 It is the caller's responsability to allocate enough memory in the buffer. to store the frames

Parameters:

sessionHandle The handle to an existing session

port The port to read from

numFrames Maximum number of frames that can be stored in the buffer

pBuffer Destination buffer for the frames

numFramesRead The actual number of frames read

Returns:

The status code

3.2.3.496 UeiDaqAPI int UeiDaqReadMIL1553BCSchedFrames (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numFrames*, tUeiMIL1553BCSchedFrame * *pBuffer*, Int32 * *numFramesRead*)

Read MIL-1553 BC Scheduler frame - i.e. major or minor entry The device associated with this data stream must support MIL-1553 It is the caller's responsability to allocate enough memory in the buffer. to store the frames

Parameters:

sessionHandle The handle to an existing session

port The port to read from

numFrames Maximum number of frames that can be stored in the buffer

pBuffer Destination buffer for the frames

numFramesRead The actual number of frames read

Returns:

The status code

3.2.3.497 UeiDaqAPI int UeiDaqReadMIL1553BCStatus (SessionHandle *sessionHandle*, Int32 *port*, tUeiMIL1553BCStatusFrame * *pBuffer*)

Read MIL-1553 RT status from the stream. The device associated with this data stream must support MIL-1553 It is the caller's responsability to allocate enough memory in the buffer. to store the frames

Parameters:

sessionHandle The handle to an existing session

port The port to read from

pBuffer Destination buffer for the frames

Returns:

The status code

3.2.3.498 UeiDaqAPI int UeiDaqReadMIL1553BMCmdFrames (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numFrames*, tUeiMIL1553BMCmdFrame * *pBuffer*, Int32 * *numFramesRead*)

Read MIL-1553 Bus Monitor protocol-parsed data The device associated with this data stream must support MIL-1553 It is the caller's responsability to allocate enough memory in the buffer. to store the frames

Parameters:

sessionHandle The handle to an existing session

port The port to read from

numFrames Maximum number of frames that can be stored in the buffer

pBuffer Destination buffer for the frames

numFramesRead The actual number of frames read

Returns:

The status code

3.2.3.499 UeiDaqAPI int UeiDaqReadMIL1553BMFrames (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numFrames*, tUeiMIL1553BMFrame * *pBuffer*, Int32 * *numFramesRead*)

Read MIL-1553 bus monitor frames from the stream. The device associated with this data stream must support MIL-1553 It is the caller's responsability to allocate enough memory in the buffer. to store the frames

Parameters:

sessionHandle The handle to an existing session

port The port to read from

numFrames Maximum number of frames that can be stored in the buffer

pBuffer Destination buffer for the frames

numFramesRead The actual number of frames read

Returns:

The status code

3.2.3.500 UeiDaqAPI int UeiDaqReadMIL1553Frames (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numFrames*, tUeiMIL1553Frame * *pBuffer*, Int32 * *numFramesRead*)

Read MIL-1553 frames from the stream. The device associated with this data stream must support MIL-1553 It is the caller's responsability to allocate enough memory in the buffer. to store the frames

Parameters:

sessionHandle The handle to an existing session

port The port to read from

numFrames Maximum number of frames that can be stored in the buffer

pBuffer Destination buffer for the frames

numFramesRead The actual number of frames read

Returns:

The status code

3.2.3.501 UeiDaqAPI int UeiDaqReadMIL1553FramesAsync (SessionHandle *sessionHandle*, Int32 *port*, Int32 *bufferSize*, tUeiMIL1553Frame * *pBuffer*, tUeiEventCallback *pEventCallback*)

Read MIL-1553 words from the stream. The device associated with this data stream must support MIL-1553 This function returns immediately, the specified callback function is called once the buffer is ready. The callback function parameter contains the number of frames actually read. It is the caller's responsability to allocate enough memory in the buffer. to store the frames

Parameters:

sessionHandle The handle to an existing session

port The port to read from

bufferSize Maximum number of frames that can be stored in the buffer

pBuffer Destination buffer for the frames

pEventCallback The callback function pointer

Returns:

The status code

3.2.3.502 UeiDaqAPI int UeiDaqReadMIL1553RTDataFrames (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numFrames*, tUeiMIL1553RTFrame * *pBuffer*, Int32 * *numFramesRead*)

Read MIL-1553 RT data area from the stream. The device associated with this data stream must support MIL-1553 It is the caller's responsability to allocate enough memory in the buffer. to store the frames

Parameters:

sessionHandle The handle to an existing session

port The port to read from

numFrames Maximum number of frames that can be stored in the buffer

pBuffer Destination buffer for the frames

numFramesRead The actual number of frames read

Returns:

The status code

3.2.3.503 UeiDaqAPI int UeiDaqReadMIL1553RTStatusFrames (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numFrames*, tUeiMIL1553RTStatusFrame * *pBuffer*, Int32 * *numFramesRead*)

Read MIL-1553 RT status from the stream. The device associated with this data stream must support MIL-1553 It is the caller's responsability to allocate enough memory in the buffer. to store the frames

Parameters:

sessionHandle The handle to an existing session

port The port to read from

numFrames Maximum number of frames that can be stored in the buffer

pBuffer Destination buffer for the frames

numFramesRead The actual number of frames read

Returns:

The status code

3.2.3.504 UeiDaqAPI int UeiDaqReadMuxADC (SessionHandle *sessionHandle*, Int32 *numVals*, double * *pBuffer*)

Read diagnostic ADC values

Parameters:

sessionHandle The handle to an existing session

numVals Number of ADC values to read

pBuffer destination buffer

Returns:

The status code

3.2.3.505 UeiDaqAPI int UeiDaqReadMuxReadRelayCounts (SessionHandle *sessionHandle*, Int32 *countsBufferSize*, Int32 * *pCountsBuffer*, Int32 * *numCountsRead*)

Read current count of times each relay has been energized

Parameters:

sessionHandle The handle to an existing session

countsBufferSize Number of relay counts to read and store in countsBuffer

pCountsBuffer Data buffer to store relay counts

numCountsRead Number of relay counts read and stored in countsBuffer

Returns:

The status code

3.2.3.506 UeiDaqAPI int UeiDaqReadMuxStatus (SessionHandle *sessionHandle*, uInt32 * *relayA*, uInt32 * *relayB*, uInt32 * *relayC*, uInt32 * *status*)

Get status of each relay on the MUX device

Parameters:

sessionHandle The handle to an existing session

relayA status of "A" relays

relayB status of "B" relays

relayC status of "C" relays

status mux status word

Returns:

The status code

3.2.3.507 UeiDaqAPI int UeiDaqReadRawData16 (SessionHandle *sessionHandle*, Int32 *timeout*, Int32 *numScans*, uInt16 * *pBuffer*)

Read 16 bits wide raw scans from the stream. It is the caller's responsability to allocate enough memory in the buffer. to store the required number of scans

Parameters:

sessionHandle The handle to an existing session

timeout The timeout determines the amount of time in ms allowed to read the specified number of scans

numScans Number of scans to read

pBuffer Destination buffer for the scans

Returns:

The status code

3.2.3.508 UeiDaqAPI int UeiDaqReadRawData16Async (SessionHandle *sessionHandle*, Int32 *numScans*, uInt16 * *pBuffer*, tUeiEventCallback *pEventCallback*)

Read 16 bits wide raw scans asynchronously from the stream. This function returns immediately, the specified callback function is called once the buffer is ready. It is the caller's responsability to allocate enough memory in the buffer. to store the required number of scans

Parameters:

sessionHandle The handle to an existing session

numScans Number of scans to read

pBuffer Destination buffer for the scans

pEventCallback The callback function pointer

Returns:

The status code

3.2.3.509 UeiDaqAPI int UeiDaqReadRawData32 (SessionHandle *sessionHandle*, Int32 *timeout*, Int32 *numScans*, uInt32 * *pBuffer*)

Read 32 bits wide raw scans from the stream. It is the caller's responsability to allocate enough memory in the buffer. to store the required number of scans

Parameters:

sessionHandle The handle to an existing session

timeout The timeout determines the amount of time in ms allowed to read the specified number of scans

numScans Number of scans to read

pBuffer Destination buffer for the scans

Returns:

The status code

3.2.3.510 UeiDaqAPI int UeiDaqReadRawData32Async (SessionHandle *sessionHandle*, Int32 *numScans*, uInt32 * *pBuffer*, tUeiEventCallback *pEventCallback*)

Read 32 bits wide raw scans asynchronously from the stream. This function returns immediately, the specified callback function is called once the buffer is ready. It is the caller's responsability to allocate enough memory in the buffer. to store the required number of scans

Parameters:

sessionHandle The handle to an existing session

numScans Number of scans to read

pBuffer Destination buffer for the scans

pEventCallback The callback function pointer

Returns:

The status code

3.2.3.511 UeiDaqAPI int UeiDaqReadScaledData (SessionHandle *sessionHandle*, Int32 *timeout*, Int32 *numScans*, f64 * *pBuffer*)

Read scaled scans from the stream. It is the caller's responsability to allocate enough memory in the buffer. to store the required number of scans

Parameters:

sessionHandle The handle to an existing session

timeout The timeout determines the amount of time in ms allowed to read the specified number of scans

numScans Number of scans to read

pBuffer Destination buffer for the scans

Returns:

The status code

3.2.3.512 UeiDaqAPI int UeiDaqReadScaledDataAsync (SessionHandle *sessionHandle*, Int32 *numScans*, f64 * *pBuffer*, tUeiEventCallback *pEventCallback*)

Read scaled scans asynchronously from the stream. This function returns immediately, the specified callback function is called once the buffer is ready. It is the caller's responsability to allocate enough memory in the buffer. to store the required number of scans

Parameters:

sessionHandle The handle to an existing session

numScans Number of scans to read

pBuffer Destination buffer for the scans

pEventCallback The callback function pointer

Returns:

The status code

3.2.3.513 UeiDaqAPI int UeiDaqReadSerialMessageTimestamped (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numBytes*, void * *pBuffer*, Int32 * *numBytesRead*)

Read message from the stream including a timestamp as the first values in the buffer. It is the caller's responsability to allocate enough memory in the buffer to store the message

Parameters:

sessionHandle The handle to an existing session

port the port to read the message from

numBytes Number of bytes to read

pBuffer Destination buffer for the messages

numBytesRead Number of bytes actually read from the port

Returns:

The status code

3.2.3.514 UeiDaqAPI int UeiDaqReadSerialMessageTimestampedAsync (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numBytes*, void * *pBuffer*, tUeiEventCallback *pEventCallback*)

Read messages asynchronously from the stream including a timestamp as the first values in the buffer. This function returns immediately, the specified callback function is called once the buffer is ready. The callback function parameter contains the number of bytes actually read. It is the caller's responsability to allocate enough memory in the buffer. to store the message

Parameters:

sessionHandle The handle to an existing session

port the port to read the message from

numBytes Number of bytes to read

pBuffer Destination buffer for the messages

pEventCallback The callback function pointer

Returns:

The status code

3.2.3.515 UeiDaqAPI int UeiDaqReadSSIMasterWords (SessionHandle *sessionHandle*, int *port*, int *grayDecoding*, Int32 *numWords*, uInt32 * *pBuffer*, Int32 * *numWordsRead*)

Read 32-bit data words from the input stream

Parameters:

sessionHandle The handle to an existing session

port The SSI master port to read from

grayDecoding true to convert gray encoded values to binary

numWords the number of words that can be stored in the destination buffer

pBuffer destination buffer

numWordsRead the actual number of frames read from the SSI port

Returns:

The status code

3.2.3.516 UeiDaqAPI int UeiDaqReadSync1PPSLockedStatus (SessionHandle *sessionHandle*, int * *isLocked*)

Parameters:

sessionHandle The handle to an existing session

isLocked Destination buffer for the status

Returns:

The status code

3.2.3.517 UeiDaqAPI int UeiDaqReadSync1PPSPTPStatus (SessionHandle *sessionHandle*, tUeiSync1PPSPTPStatus * *pBuffer*)

Read PTP status from the stream.

Parameters:

sessionHandle The handle to an existing session

pBuffer Destination buffer for the status

Returns:

The status code

3.2.3.518 UeiDaqAPI int UeiDaqReadSync1PPSPTPUTCTime (SessionHandle *sessionHandle*, tUeiPTPTime * *pBuffer*)

Read PTP UTC time from the stream.

Parameters:

sessionHandle The handle to an existing session

pBuffer Destination buffer for the UTC time

Returns:

The status code

3.2.3.519 UeiDaqAPI int UeiDaqReadSync1PPSStatus (SessionHandle *sessionHandle*, tUeiSync1PPSStatus * *pBuffer*)

Read 1PPS synchronization status from the stream.

Parameters:

sessionHandle The handle to an existing session

pBuffer Destination buffer for the messages

Returns:

The status code

3.2.3.520 UeiDaqAPI int UeiDaqReadVRADCData (SessionHandle *sessionHandle*, Int32 *channel*, Int32 *numValues*, double * *pBuffer*, Int32 * *numValuesRead*)

Read ADC FIFO data, this is used as a diagnostic to visualize the shape of the signal as it is acquired on the VR input device. There is one FIFO for each pair of channel. this function reads the ADC data for both even and odd channels

Parameters:

sessionHandle The handle to an existing session

channel The channel to read from

numValues number of values to read

pBuffer destination buffer for even and odd channel ADC data

numValuesRead Number of values actually read

Returns:

The status code

3.2.3.521 UeiDaqAPI int UeiDaqReadVRADCStatus (SessionHandle *sessionHandle*, Int32 *channel*, uInt32 * *pStatus*)

Read ADC status data, this is used as a diagnostic to troubleshoot VR input

Parameters:

sessionHandle The handle to an existing session

channel The channel to read from

pStatus destination buffer for even and odd channel ADC data

Returns:

The status code

3.2.3.522 UeiDaqAPI int UeiDaqReadVRData (SessionHandle *sessionHandle*, Int32 *channel*, Int32 *numValues*, tUeiVRData * *pBuffer*, Int32 * *numValuesRead*)

Read variable reluctance data. Each value comes as a structure that includes the VR sensor velocity (teeth/sec), position, teeth count and timestamp

Parameters:

sessionHandle The handle to an existing session

channel The channel to read from

numValues number of values to read

pBuffer destination buffer

numValuesRead Number of values actually read

Returns:

The status code

3.2.3.523 UeiDaqAPI int UeiDaqReadVRFifoData (SessionHandle *sessionHandle*, Int32 *channel*, Int32 *numValues*, uInt32 * *pBuffer*, Int32 * *numValuesRead*)

Variable reluctance FIFO data can be used to calculate a map of the inter-tooth delays around the encoder wheel. this is useful to calculate acceleration.

Parameters:

sessionHandle The handle to an existing session

channel The channel to read from

numValues number of values to read

pBuffer destination buffer

numValuesRead Number of values actually read

Returns:

The status code

3.2.3.524 UeiDaqAPI void UeiDaqReloadDrivers ()

Un-load/re-load driver plugins

3.2.3.525 UeiDaqAPI int UeiDaqResetCircuitBreaker (SessionHandle *sessionHandle*, Int32 *port*, uInt32 *mask*)

Reset a circuit breaker, in case it was tripped.

Parameters:

sessionHandle The handle to an existing session

port the port to set breaker on

mask the mask specifying which circuit breaker to reset (1 to reset, 0 to leave alone)

Returns:

The status code

3.2.3.526 UeiDaqAPI int UeiDaqResetDevice (DeviceHandle *deviceHandle*)

Executes a hardware reset on the device. To reboot a PowerDNA or PowerDNR unit call this method on the CPU device (device 14).

Parameters:

deviceHandle The handle to an existing device

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.527 UeiDaqAPI int UeiDaqResumeSession (SessionHandle *sessionHandle*, int *port*)

Pause the specified messaging port.

Parameters:

sessionHandle The handle to an existing session

port The port to resume

Returns:

The status code

3.2.3.528 UeiDaqAPI int UeiDaqSendSerialBreak (SessionHandle *sessionHandle*, Int32 *port*, uInt32 *durationMs*)

Send serial break

Parameters:

sessionHandle The handle to an existing session

port the port to write the message to

durationMs The duration of the break in milliseconds

Returns:

The status code

3.2.3.529 UeiDaqAPI int UeiDaqSetAccelCouplingType (ChannelHandle *channelHandle*, tUeiCoupling *coupling*)

Configure the channel in AC or DC coupling.

Parameters:

channelHandle The handle to an existing accelerometer channel

coupling The coupling type

Returns:

The status code.

See also:

UeiDaqGetAccelCouplingType (p. 755)

3.2.3.530 UeiDaqAPI int UeiDaqSetAccelExcitationCurrent (ChannelHandle *channelHandle*, double *excCurrent*)

Set the excitation current in mAmps

Parameters:

channelHandle The handle to an existing accelerometer channel

excCurrent The new excitation current

Returns:

The status code.

See also:

UeiDaqGetAccelExcitationCurrent (p. 755)

3.2.3.531 UeiDaqAPI int UeiDaqSetAccelHighExcitationComparator (ChannelHandle *channelHandle*, double *highComparator*)

The excitation current can be measured back and trigger an alarm when it is out of range (due to an open connection or a short). It is measured as a voltage which depends on the sensor resistance and the programmed excitation current. The alarm state is made visible with the LED color on the terminal block Green means that the measured excitation is within range, red means that the connection with the sensor is open and blinking red means that the connection with the sensor is shorted.

Parameters:

channelHandle The handle to an existing accelerometer channel

highComparator the new high excitation comparator voltage

Returns:

The status code.

See also:

UeiDaqGetAccelHighExcitationComparator (p. 756)

3.2.3.532 UeiDaqAPI int UeiDaqSetAccelLowExcitationComparator (ChannelHandle *channelHandle*, double *lowComparator*)

The excitation current can be measured back and trigger an alarm when it is out of range (due to an open connection or a short). It is measured as a voltage which depends on the sensor resistance and the programmed excitation current. The alarm state is made visible with the LED color on the terminal block Green means that the measured excitation is within range, red means that the connection with the sensor is open and blinking red means that the connection with the sensor is shorted.

Parameters:

channelHandle The handle to an existing accelerometer channel

lowComparator the new low excitation comparator voltage

Returns:

The status code.

See also:

UeiDaqGetAccelLowExcitationComparator (p. 756)

3.2.3.533 UeiDaqAPI int UeiDaqSetAccelSensorSensitivity (ChannelHandle *channelHandle*, double *sensitivity*)

Set the sensor sensitivity, it usually is in mVolts/g Independently of the unit, the voltage read will be converted to mV and divided by the sensitivity

Parameters:

channelHandle The handle to an existing accelerometer channel

sensitivity The new sensitivity

Returns:

The status code.

See also:

UeiDaqGetAccelSensorSensitivity (p. 757)

3.2.3.534 UeiDaqAPI int UeiDaqSetAIChannelGain (ChannelHandle *channelHandle*, f64 *gain*)

Overrides gain calcualted from input range

Parameters:

channelHandle The handle to an existing channel

gain the new gain

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.535 UeiDaqAPI int UeiDaqSetAIChannelInputMode (ChannelHandle *channelHandle*, tUeiAIChannelInputMode *mode*)

Set the input mode of the channel, possible values are "differential" or "single-ended".

Parameters:

channelHandle The handle to an existing channel

mode The input mode.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.536 UeiDaqAPI int UeiDaqSetAIChannelMaximum (ChannelHandle *channelHandle*, f64 *maximum*)

Set the maximum expected value for the signal connected to the channel.

Parameters:

channelHandle The handle to an existing channel

maximum The maximum value.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.537 UeiDaqAPI int UeiDaqSetAIChannelMinimum (ChannelHandle *channelHandle*, f64 *minimum*)

Set the minimum expected value for the signal connected to the channel.

Parameters:

channelHandle The handle to an existing channel

minimum The minimum value.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.538 UeiDaqAPI int UeiDaqSetAIChannelMuxPos (ChannelHandle *channelHandle*, tUeiAIChannelMuxPos *muxPos*)

Set the mux position mode for devices with self-test capability

Parameters:

channelHandle The handle to an existing channel

muxPos The new mux position mode

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.539 UeiDaqAPI int UeiDaqSetAIVExBridgeCompletionSetting (ChannelHandle *channelHandle*, double *setting*)

Set the offset nulling setting used to program the nulling circuitry. Set it to 0.0 to automatically perform offset nulling next time the session is started. Make sure no strain is applied on the bridge before nulling the offset.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

setting The setting value used to program the bridge completion circuitry.

Returns:

The status code.

See also:

UeiDaqGetAIVExBridgeCompletionSetting (p. 760)

3.2.3.540 UeiDaqAPI int UeiDaqSetAIVExBridgeType (ChannelHandle *channelHandle*, tUeiSensorBridgeType *type*)

Set the bridge type for the sensor connected to this channel.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

type The bridge type

Returns:

The status code.

See also:

UeiDaqGetAIVExBridgeType (p. 760)

3.2.3.541 UeiDaqAPI int UeiDaqSetAIVExExcitationFrequency (ChannelHandle *channelHandle*, double *fex*)

Set the excitation frequency for this channel.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

fex The excitation frequency

Returns:

The status code.

See also:

UeiDaqGetAIVExExcitationFrequency (p. 760)

3.2.3.542 UeiDaqAPI int UeiDaqSetAIVExExcitationVoltage (ChannelHandle *channelHandle*, double *vex*)

Set the excitation voltage for this channel.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

vex The excitation voltage

Returns:

The status code.

See also:

UeiDaqGetAIVExExcitationVoltage (p. 761)

3.2.3.543 UeiDaqAPI int UeiDaqSetAIVExGainAdjustmentFactor (ChannelHandle *channelHandle*, double *gaf*)

Setting the gain adjustment factor is part of the shunt calibration procedure: You must first program the shunt resistance, engage the the shunt resistor and measure some values. You can then calculate the gain adjustment factor with the formula: gaf = Vex∗(Rg/(4∗Rs + 2∗Rg))/Vin Vex: excitation voltage Vin: measured voltage Rg: Resistance of the strain gages. Rs: Resistance of the shunt.

Once the gain adjustment factor is set, the framework automatically uses it to scale measurements to engennering unit.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

gaf The new gain adjustment factor.

Returns:

The status code.

See also:

UeiDaqGetAIVExGainAdjustmentFactor (p. 761)

3.2.3.544 UeiDaqAPI int UeiDaqSetAIVExOffsetNullingSetting (ChannelHandle *channelHandle*, double *setting*)

Set the offset nulling setting used to program the nulling circuitry. Set it to 0.0 to automatically perform offset nulling next time the session is started. Make sure no strain is applied on the bridge before nulling the offset.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

setting The setting value used to program the offset nulling circuitry.

Returns:

The status code.

See also:

UeiDaqGetAIVExOffsetNullingSetting (p. 762)

3.2.3.545 UeiDaqAPI int UeiDaqSetAIVExScalingWithExcitation (ChannelHandle *channelHandle*, int *scaleWithExcitation*)

Specifies whether the acquired data will be divided by the excitation voltage before it is returned.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

scaleWithExcitation The scaling mode

Returns:

The status code.

See also:

UeiDaqGetAIVExScalingWithExcitation (p. 762)

3.2.3.546 UeiDaqAPI int UeiDaqSetAIVExShuntLocation (ChannelHandle *channelHandle*, tUeiWheatstoneBridgeBranch *shuntedBranch*)

Specifies the wheatstone bridge branch to shunt.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

shuntedBranch The bridge branch to shunt.

Returns:

The status code.

See also:

UeiDaqGetAIVExShuntLocation (p. 762)

3.2.3.547 UeiDaqAPI int UeiDaqSetAIVExShuntResistance (ChannelHandle *channelHandle*, double *resistance*)

Specifies the resistance to program to the shunt resistor in Ohms. NOTE: Don't use this value to calculate the gain adjustment factor, Use the value returned by GetActualShuntResitance() which takes into account the resistance of other non-programmable parts in the shunt circuit.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

resistance The resistance to program on the shunt resistor.

Returns:

The status code.

See also:

UeiDaqGetAIVExShuntResistance (p. 763) **UeiDaqGetAIVExActualShuntResistance** (p. 759)

3.2.3.548 UeiDaqAPI int UeiDaqSetAIVExWiringScheme (ChannelHandle *channelHandle*, tUeiWiringScheme *wiring*)

Specifies the wiring scheme used to connect the strain gage or load cell to the channel.

Parameters:

channelHandle The handle to an existing voltage with excitation channel

wiring The new wiring scheme.

Returns:

The status code.

See also:

UeiDaqGetAIVExWiringScheme (p. 763)

3.2.3.549 UeiDaqAPI int UeiDaqSetAOChannelDefaultValue (ChannelHandle *channelHandle*, f64 *defaultVal*)

Set the default value.

Parameters:

channelHandle The handle to an existing channel

defaultVal the default value.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.550 UeiDaqAPI int UeiDaqSetAOChannelMaximum (ChannelHandle *channelHandle*, f64 *maximum*)

Set the maximum value for the generated signal.

Parameters:

channelHandle The handle to an existing channel

maximum The maximum value.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.551 UeiDaqAPI int UeiDaqSetAOChannelMinimum (ChannelHandle *channelHandle*, f64 *minimum*)

Set the minimum value for the generated signal.

Parameters:

channelHandle The handle to an existing channel

minimum The minimum value.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.552 UeiDaqAPI int UeiDaqSetAOProtectedADCChannel (ChannelHandle *channelHandle*, int *index*, tUeiAODiagChannel *adcChannel*)

Each AO channel comes with up to five diagnostic channels. Set the diagnostic channel associated with the specified index.

Parameters:

channelHandle The handle to an existing channel

index The 0 based index of the diagnostic channel

adcChannel The new diagnostic channel at specified index

Returns:

The status code

3.2.3.553 UeiDaqAPI int UeiDaqSetAOProtectedAutoRetry (ChannelHandle *channelHandle*, int *autoRetry*)

The auto retry setting specifies wheter the device should attempt to close the circuit after it was opened because of an over or under limit condition. If it is set to true the device will try to close the circuit at a rate set by the autoRetryRate setting.

Parameters:

channelHandle The handle to an existing channel

autoRetry true to turn auto-retry on, false to turn it off.

Returns:

The status code

3.2.3.554 UeiDaqAPI int UeiDaqSetAOProtectedAutoRetryRate (ChannelHandle *channelHandle*, double *autoRetryRate*)

Specifies how often the device will attemp to close a circuit that was opened because of an over or under limit condition.

Parameters:

channelHandle The handle to an existing channel

autoRetryRate The new number of retries per second.

Returns:

The status code

3.2.3.555 UeiDaqAPI int UeiDaqSetAOProtectedCircuitBreakerHighLimit (ChannelHandle *channelHandle*, int *index*, double *highLimit*)

Each circuit-breaker can monitor one diagnostic channel. Set the maximum current/volt/temperature allowed on the specified channel. The circuit will open if more than the minimum value is monitored.

Parameters:

channelHandle The handle to an existing channel

index The 0 based index of the circuit breaker

highLimit The new high limit.

Returns:

The status code

3.2.3.556 UeiDaqAPI int UeiDaqSetAOProtectedCircuitBreakerLowLimit (ChannelHandle *channelHandle*, int *index*, double *lowLimit*)

Each circuit-breaker can monitor one diagnostic channel. Set the minimum current/volt/temperature allowed on the specified channel. The circuit will open if less than the minimum value is monitored.

Parameters:

channelHandle The handle to an existing channel

index The 0 based index of the circuit breaker

lowLimit The new low limit.

Returns:

The status code

3.2.3.557 UeiDaqAPI int UeiDaqSetAOProtectedDACMode (ChannelHandle *channelHandle*, tUeiAODACMode *mode*)

Each channel uses two DACs that can be enabled independantly.

Parameters:

channelHandle The handle to an existing channel

mode The new DAC mode.

Returns:

The status code

3.2.3.558 UeiDaqAPI int UeiDaqSetAOProtectedMeasurementRate (ChannelHandle *channelHandle*, double *measurementRate*)

Set the rate at which the diagnostic channels are monitored. This rate determines how fast the device react when an under or over limit condition occurs.

Parameters:

channelHandle The handle to an existing channel

measurementRate The new circuit breaker measurement rate.

Returns:

The status code

3.2.3.559 UeiDaqAPI int UeiDaqSetAOProtectedOverUnderCount (ChannelHandle *channelHandle*, uInt32 *overUnderCount*)

Specifies number of consecutive over/under limit diagnostic readings that must occur in order to trip breaker.

Parameters:

channelHandle The handle to an existing channel

overUnderCount The new maximum number of over/under readings.

Returns:

The status code

3.2.3.560 UeiDaqAPI int UeiDaqSetAOShortOpenCircuit (SessionHandle *sessionHandle*, uInt32 *openBitmask*, uInt32 *shortBitmask*, uInt32 *shortDurationUs*)

Configure short/open circuit simulation using bit masks Set bit to 1 to sohrt or open the corresponding channel If a channel is commanded to be both open and shorted it will be shorted.

Parameters:

sessionHandle The handle to an existing session

openBitmask the bitmask controlling which channel should be open

shortBitmask the bitmask controlling which channel should be shorted

shortDurationUs the maximum of time that shorted channels will stay shorted

Returns:

The status code

3.2.3.561 UeiDaqAPI int UeiDaqSetAOWaveformMainDACClockSource (ChannelHandle *channelHandle*, tUeiAOWaveformClockSource *source*)

Set the source of the clock used by the main DAC.

Parameters:

channelHandle The handle to an existing channel

source the new main DAC clock source.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.562 UeiDaqAPI int UeiDaqSetAOWaveformMainDACClockSync (ChannelHandle *channelHandle*, tUeiAOWaveformClockSync *sync*)

Set the output where the clock used by the main DAC will be routed.

Parameters:

channelHandle The handle to an existing channel

sync the new main DAC clock synchronization.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.563 UeiDaqAPI int UeiDaqSetAOWaveformMainDACTriggerSource (ChannelHandle *channelHandle*, tUeiAOWaveformTriggerSource *source*)

Set the source of the trigger used by the main DAC.

Parameters:

channelHandle The handle to an existing channel

source the new main DAC trigger source.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.564 UeiDaqAPI int UeiDaqSetAOWaveformOffsetDACClockSource (ChannelHandle *channelHandle*, tUeiAOWaveformOffsetClockSource *source*)

Set the source of the clock used by the offset DAC.

Parameters:

channelHandle The handle to an existing channel

source the new offset DAC clock source.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.565 UeiDaqAPI int UeiDaqSetAOWaveformOffsetDACTriggerSource (ChannelHandle *channelHandle*, tUeiAOWaveformOffsetTriggerSource *source*)

Set the source of the trigger used by the offset DAC.

Parameters:

channelHandle The handle to an existing channel

source the new offset DAC trigger source.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.566 UeiDaqAPI int UeiDaqSetARINCInputParity (ChannelHandle *channelHandle*, tUeiARINCPortParity *parity*)

Set the parity used to detect transmission errors. The parity bit of each ARINC frame is set to 0 or 1 so that the number of bits set to 1 matches the specified parity.

Parameters:

channelHandle The handle to an existing ARINC-429 input channel

parity The ARINC port parity

Returns:

The status code.

See also:

UeiDaqGetARINCInputParity (p. 769)

3.2.3.567 UeiDaqAPI int UeiDaqSetARINCInputSDIFilterMask (ChannelHandle *channelHandle*, uInt32 *SDIFilterMask*)

Set the SDI filter mask, only bits 0 and 1 are meaningful. ARINC frames whose SDI bits don't match the SDI mask are rejected.

Parameters:

channelHandle The handle to an existing ARINC-429 input channel

SDIFilterMask The new SDI filter mask

Returns:

The status code.

See also:

UeiDaqGetARINCInputSDIFilterMask (p. 770)

3.2.3.568 UeiDaqAPI int UeiDaqSetARINCInputSpeed (ChannelHandle *channelHandle*, tUeiARINCPortSpeed *bitsPerSecond*)

Set the number of bits transmitted per second.

Parameters:

channelHandle The handle to an existing ARINC-429 input channel

bitsPerSecond The ARINC port speed

Returns:

The status code.

See also:

UeiDaqGetARINCInputSpeed (p. 770)

3.2.3.569 UeiDaqAPI int UeiDaqSetARINCOutputFIFORate (ChannelHandle *channelHandle*, double *rate*)

Sets TX FIFO rate.

Parameters:

channelHandle The handle to an existing ARINC-429 output channel

rate the new TX FIFO rate

Returns:

The status code.

See also:

UeiDaqGetARINCOutputFIFORate (p. 770)

3.2.3.570 UeiDaqAPI int UeiDaqSetARINCOutputParity (ChannelHandle *channelHandle*, tUeiARINCPortParity *parity*)

Set the parity used to detect transmission errors. The parity bit of each ARINC frame is set to 0 or 1 so that the number of bits set to 1 matches the specified parity.

Parameters:

channelHandle The handle to an existing ARINC-429 output channel

parity The ARINC port parity

Returns:

The status code.

See also:

UeiDaqGetARINCOutputParity (p. 771)

3.2.3.571 UeiDaqAPI int UeiDaqSetARINCOutputSchedulerRate (ChannelHandle *channelHandle*, double *rate*)

Sets scheduler rate.

Parameters:

channelHandle The handle to an existing ARINC-429 output channel

rate the new scheduler rate

Returns:

The status code.

See also:

UeiDaqGetARINCOutputSchedulerRate (p. 772)

3.2.3.572 UeiDaqAPI int UeiDaqSetARINCOutputSchedulerType (ChannelHandle *channelHandle*, tUeiARINCSchedulerType *type*)

Sets scheduler type.

Parameters:

channelHandle The handle to an existing ARINC-429 output channel

type the new scheduler type

Returns:

The status code.

See also:

UeiDaqGetARINCOutputSchedulerType (p. 772)

3.2.3.573 UeiDaqAPI int UeiDaqSetARINCOutputSpeed (ChannelHandle *channelHandle*, tUeiARINCPortSpeed *bitsPerSecond*)

Set the number of bits transmitted per second.

Parameters:

channelHandle The handle to an existing ARINC-429 output channel

bitsPerSecond The ARINC port speed

Returns:

The status code.

See also:

UeiDaqGetARINCOutputSpeed (p. 772)

3.2.3.574 UeiDaqAPI int UeiDaqSetARINCTransmitPage (SessionHandle *sessionHandle*, Int32 *port*, int *immediate*, uInt32 *writeMask*, uInt32 *txMask*)

Major/Minor scheduler features two copies of the scheduler entries. Page 0 and page 1. This call specifies which channel should be written to and which channel should be transmitting. This is designed to configure or update a page while the other is being transmitted. But it is still possible to both write and transmit a channel at the same time.

Parameters:

sessionHandle The handle to an existing session

port the port to write the message to

immediate Specifies when the page configuration will be updated (true to update immediately, false to wait for the next major frame clock)

writeMask Bit mask specifying the writing page for each channel (bit set to 0 for page 0 or 1 for page 1)

txMask Bit mask specifying the transmitting page for each channel (bit set to 0 for page 0 or 1 for page 1)

Returns:

The status code

3.2.3.575 UeiDaqAPI int UeiDaqSetCANPortAcceptanceCode (ChannelHandle *channelHandle*, uInt32 *code*)

The acceptance code is used to filter incoming frames. The arbitration ID bits selected by the mask are compared to the code and the frame is rejected if there is any difference. If (mask XOR id) AND code == 0 the frame is accepted

Parameters:

channelHandle The handle to an existing CAN port

code The acceptance code

Returns:

The status code.

See also:

UeiDaqGetCANPortAcceptanceCode (p. 773)

3.2.3.576 UeiDaqAPI int UeiDaqSetCANPortAcceptanceMask (ChannelHandle *channelHandle*, uInt32 *mask*)

The acceptance mask is used to filter incoming frames. The mask selects which bits within arbitration ID will be used for filtering.

Parameters:

channelHandle The handle to an existing CAN port

mask The acceptance mask

Returns:

The status code.

See also:

UeiDaqGetCANPortAcceptanceMask (p. 774)

3.2.3.577 UeiDaqAPI int UeiDaqSetCANPortBitTimingRegisters (ChannelHandle *channelHandle*, uInt16 *btrValues*)

Set the values of the bit timing registers from the NXP SJA1000 CAN controller chip. BTR0 and BTR1 set the baud rate for the CAN port. For information on CAN bit timing registers, refer to the datasheet of the NXP SJA1000 CAN controller, available for download on www.nxp.com.

Parameters:

channelHandle The handle to an existing CAN port

btrValues The new bit timing regiters, BTR0 is LSB and BTR1 is MSB.

Returns:

The status code.

3.2.3.578 UeiDaqAPI int UeiDaqSetCANPortFrameFormat (ChannelHandle *channelHandle*, tUeiCANFrameFormat *frameFormat*)

Set the format of frames transmitted by this port (CAN frames can be basic or extended).

Parameters:

channelHandle The handle to an existing CAN port

frameFormat The Frame format

Returns:

The status code.

See also:

UeiDaqGetCANPortFrameFormat (p. 776)

3.2.3.579 UeiDaqAPI int UeiDaqSetCANPortMode (ChannelHandle *channelHandle*, tUeiCANPortMode *mode*)

Set the CAN port operation mode. Possible values are normal and passive

Parameters:

channelHandle The handle to an existing CAN port

mode The operation mode

Returns:

The status code.

See also:

UeiDaqGetCANPortMode (p. 776)

3.2.3.580 UeiDaqAPI int UeiDaqSetCANPortSpeed (ChannelHandle *channelHandle*, tUeiCANPortSpeed *bitsPerSecond*)

Set the number of data bits transmitted per second.

Parameters:

channelHandle The handle to an existing CAN port

bitsPerSecond The port speed

Returns:

The status code.

See also:

UeiDaqGetCANPortSpeed (p. 777)

3.2.3.581 UeiDaqAPI int UeiDaqSetChannelAliasName (ChannelHandle *channelHandle*, char * *aliasName*)

Set the alias name of the channel

Parameters:

channelHandle The handle to an existing channel

aliasName A string that contains the new alias name.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.582 UeiDaqAPI int UeiDaqSetChannelIndex (ChannelHandle *channelHandle*, int *index*)

Set the index of the channel in the session's channel list

Parameters:

channelHandle The handle to an existing channel

index The channel index.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.583 UeiDaqAPI int UeiDaqSetChannelResourceName (ChannelHandle *channelHandle*, char * *resName*)

Set the channel URL. Ex: pwrdaq://Dev1/ai0..

Parameters:

channelHandle The handle to an existing channel

resName A string that contains the resource name.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.584 UeiDaqAPI int UeiDaqSetCIChannelCounterGate (ChannelHandle *channelHandle*, tUeiCounterGate *gate*)

Set the signal that specify whether the counter/timer is on or off

Parameters:

channelHandle The handle to an existing channel

gate The gate.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.585 UeiDaqAPI int UeiDaqSetCIChannelCounterMode (ChannelHandle *channelHandle*, tUeiCounterMode *mode*)

Set the mode that specify whether the session counts events, measures a pulse width or measures a period on the signal configured as the source

Parameters:

channelHandle The handle to an existing channel

mode The new mode.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.586 UeiDaqAPI int UeiDaqSetCIChannelCounterSource (ChannelHandle *channelHandle*, tUeiCounterSource *source*)

Set the source of the signal counted or used as a timebase

Parameters:

channelHandle The handle to an existing channel

source The source.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.587 UeiDaqAPI int UeiDaqSetCIChannelInverted (ChannelHandle *channelHandle*, Int32 *inverted*)

Specifies whether the signal at the source is inverted before performing the counting operation

Parameters:

channelHandle The handle to an existing channel

inverted The inverted setting.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.588 UeiDaqAPI int UeiDaqSetCIGateMode (ChannelHandle *channelHandle*, tUeiCounterGateMode *gateMode*)

Set the gate mode which determines whether the counter/timer will run continuously once the gate is asserted

Parameters:

channelHandle The handle to an existing channel

gateMode the new gate mode.

Returns:

The status code

See also:

UeiDaqGetCIGateMode (p. 780)

3.2.3.589 UeiDaqAPI int UeiDaqSetCIMinimumGatePulseWidth (ChannelHandle *channelHandle*, double *minWidth*)

Set the minimum pulse width in ms the counter recognizes at its gate. Any pulse whose width is smaller will be ignored.

Parameters:

channelHandle The handle to an existing channel

minWidth the minimum pulse width

Returns:

The status code

See also:

UeiDaqGetCIMinimumGatePulseWidth (p. 781)

3.2.3.590 UeiDaqAPI int UeiDaqSetCIMinimumSourcePulseWidth (ChannelHandle *channelHandle*, double *minWidth*)

Set the minimum pulse width in ms the counter recognizes at its source. Any pulse whose width is smaller will be ignored.

Parameters:

channelHandle The handle to an existing channel

minWidth the minimum pulse width

Returns:

The status code

See also:

UeiDaqGetCIMinimumSourcePulseWidth (p. 781)

3.2.3.591 UeiDaqAPI int UeiDaqSetCIPeriodCount (ChannelHandle *channelHandle*, Int32 *periodCount*)

Set the number of period(s) to measure. An average measurement is returned.

Parameters:

channelHandle The handle to an existing channel

periodCount the new period count.

Returns:

The status code

See also:

UeiDaqGetCIPeriodCount (p. 781)

3.2.3.592 UeiDaqAPI int UeiDaqSetCITimebaseDivider (ChannelHandle *channelHandle*, Int32 *divider*)

Set the divider used to scale the timebase speed.

Parameters:

channelHandle The handle to an existing channel

divider the new divider value.

Returns:

The status code

See also:

UeiDaqGetCITimebaseDivider (p. 782)

3.2.3.593 UeiDaqAPI int UeiDaqSetCOChannelCounterGate (ChannelHandle *channelHandle*, tUeiCounterGate *gate*)

Set the signal that specify whether the counter/timer is on or off

Parameters:

channelHandle The handle to an existing channel

gate The gate.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.594 UeiDaqAPI int UeiDaqSetCOChannelCounterMode (ChannelHandle *channelHandle*, tUeiCounterMode *mode*)

Set the mode that specify whether the session generates a single pulse or a pulse train on the counter output

Parameters:

channelHandle The handle to an existing channel

mode The new mode.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.595 UeiDaqAPI int UeiDaqSetCOChannelCounterSource (ChannelHandle *channelHandle*, tUeiCounterSource *source*)

Set the source of the signal used as a timebase

Parameters:

channelHandle The handle to an existing channel

source The source.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.596 UeiDaqAPI int UeiDaqSetCOChannelInverted (ChannelHandle *channelHandle*, Int32 *inverted*)

Specifies whether the signal at the source is inverted before performing the timing operation

Parameters:

channelHandle The handle to an existing channel

inverted The inverted setting.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.597 UeiDaqAPI int UeiDaqSetCOChannelTick1 (ChannelHandle *channelHandle*, Int32 *tick1*)

Specifies how long the output stays low.

Parameters:

channelHandle The handle to an existing channel

tick1 The number of ticks of the signal connected at the source.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.598 UeiDaqAPI int UeiDaqSetCOChannelTick2 (ChannelHandle *channelHandle*, Int32 *tick2*)

Specifies how long the output stays high.

Parameters:

channelHandle The handle to an existing channel

tick2 The number of ticks of the signal connected at the source.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.599 UeiDaqAPI int UeiDaqSetCOGateMode (ChannelHandle *channelHandle*, tUeiCounterGateMode *gateMode*)

Set the gate mode which determines whether the counter/timer will run continuously once the gate is asserted

Parameters:

channelHandle The handle to an existing channel

gateMode the new gate mode.

Returns:

The status code

See also:

UeiDaqGetCOGateMode (p. 784)

3.2.3.600 UeiDaqAPI int UeiDaqSetCOMinimumGatePulseWidth (ChannelHandle *channelHandle*, double *minWidth*)

Set the minimum pulse width in ms the counter recognizes at its gate. Any pulse whose width is smaller will be ignored.

Parameters:

channelHandle The handle to an existing channel

minWidth the minimum pulse width

Returns:

The status code

See also:

UeiDaqGetCOMinimumGatePulseWidth (p. 785)

3.2.3.601 UeiDaqAPI int UeiDaqSetCOMinimumSourcePulseWidth (ChannelHandle *channelHandle*, double *minWidth*)

Set the minimum pulse width in ms the counter recognizes at its source. Any pulse whose width is smaller will be ignored.

Parameters:

channelHandle The handle to an existing channel

minWidth the minimum pulse width

Returns:

The status code

See also:

UeiDaqGetCOMinimumSourcePulseWidth (p. 785)

3.2.3.602 UeiDaqAPI int UeiDaqSetCONumberOfPulses (ChannelHandle *channelHandle*, Int32 *numberOfPulses*)

Set the number of pulses to generate (-1 for continuous)

Parameters:

channelHandle The handle to an existing channel

numberOfPulses the new number of pulses.

Returns:

The status code

See also:

GetNumberOfPulses

3.2.3.603 UeiDaqAPI int UeiDaqSetCOTimebaseDivider (ChannelHandle *channelHandle*, Int32 *divider*)

Set the divider used to scale the timebase speed.

Parameters:

channelHandle The handle to an existing channel

divider the new divider value.

Returns:

The status code

See also:

UeiDaqGetCOTimebaseDivider (p. 786)

3.2.3.604 UeiDaqAPI int UeiDaqSetDataStreamBurst (DataStreamHandle *dataStreamHandle*, Int32 *burst*)

Specifies whether the input device will acquire all the requested data in its internal memory before transferring it to the host PC.

Parameters:

dataStreamHandle The handle to an existing data stream

burst the new burst setting value.

Returns:

The status code

See also:

UeiDaqGetDataStreamBurst (p. 786)

3.2.3.605 UeiDaqAPI int UeiDaqSetDataStreamNumberOfFrames (DataStreamHandle *dataStreamHandle*, Int32 *numFrames*)

Set the number of frames. Each frame is accessed independently by the device.

Parameters:

dataStreamHandle The handle to an existing data stream

numFrames The number of frames.

Returns:

The status code

See also:

UeiDaqGetDataStreamHandle (p. 787).

3.2.3.606 UeiDaqAPI int UeiDaqSetDataStreamNumberOfScans (DataStreamHandle *dataStreamHandle*, Int32 *numScans*)

Set the number of scans to read at a time

Parameters:

dataStreamHandle The handle to an existing data stream

numScans The number of scans.

Returns:

The status code

See also:

UeiDaqGetDataStreamHandle (p. 787)

3.2.3.607 UeiDaqAPI int UeiDaqSetDataStreamOffset (DataStreamHandle *dataStreamHandle*, Int32 *offset*)

Move the position of the next scan to read or write. The offset is relative to the reference specified with SetRelativeTo(). Use a negative offset to read or write scans below the reference position.

Parameters:

dataStreamHandle The handle to an existing data stream

offset The new offset.

Returns:

The status code

See also:

UeiDaqGetDataStreamHandle (p. 787).

3.2.3.608 UeiDaqAPI int UeiDaqSetDataStreamOverUnderRun (DataStreamHandle *dataStreamHandle*, Int32 *overunderrun*)

For an input session, Specifies whether the device overwrite acquired data if it was not read fast enough. For an output session, Specifies whether the device regenerate previously generated data if it didn't receive new data fast enough. If overUnderRun is set to 1, the framework ignore the overrun or underrun condition. If it is set to 0 an exception is thrown.

Parameters:

dataStreamHandle The handle to an existing data stream

overunderrun The overrun/underrun setting.

Returns:

The status code

See also:

UeiDaqGetDataStreamHandle (p. 787).

3.2.3.609 UeiDaqAPI int UeiDaqSetDataStreamRegenerate (DataStreamHandle *dataStreamHandle*, Int32 *regenerate*)

Specifies whether the output device will continuously regenerate the first buffer it received.

Parameters:

dataStreamHandle The handle to an existing data stream

regenerate The regenerate setting.

Returns:

The status code

See also:

UeiDaqGetDataStreamHandle (p. 787).

3.2.3.610 UeiDaqAPI int UeiDaqSetDataStreamRelativeTo (DataStreamHandle *dataStreamHandle*, tUeiDataStreamRelativeTo *relativeTo*)

Set the position to use as a reference in the framework's internal buffer

Parameters:

dataStreamHandle The handle to an existing data stream

relativeTo The new reference.

Returns:

The status code

See also:

UeiDaqGetDataStreamHandle (p. 787).

3.2.3.611 UeiDaqAPI int UeiDaqSetDataStreamScanSize (DataStreamHandle *dataStreamHandle*, Int32 *scanSize*)

Set the size of a scan, usually equal to the number of channels in the session's channel list.

Parameters:

dataStreamHandle The handle to an existing data stream

scanSize The number of scans.

Returns:

The status code

See also:

UeiDaqGetDatastreamHandle.

3.2.3.612 UeiDaqAPI int UeiDaqSetDeviceTimeout (DeviceHandle *deviceHandle*, Int32 *timeout*)

Set the maximum amount of time (in ms) for a low-level access request to complete. Set to -1 to use default value (depends on hardware)

Parameters:

deviceHandle The handle to an existing device

timeout the timeout

Returns:

The status code

See also:

UeiDaqGetDeviceTimeout (p. 805), **UeiDaqGetDeviceHandle** (p. 795)

3.2.3.613 UeiDaqAPI int UeiDaqSetDeviceWatchDogCommand (DeviceHandle *deviceHandle*, tUeiWatchDogCommand *cmd*, int *timeout*)

Enable/Disable watchdog and configure mode. This is only supported on PowerDNA/DNR/DNF CPU devices (device 14)

Parameters:

deviceHandle The handle to a watchdog capable device

cmd the watchdog command

timeout the watchdog expiration timeout

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.614 UeiDaqAPI int UeiDaqSetDIChannelEdgeMask (ChannelHandle *channelHandle*, uInt32 *mask*, tUeiDigitalEdge *edgeType*)

Set the mask used to select wich digital line will be used to sense edges. When bit x is set to 1, line x is used.

Parameters:

channelHandle The handle to an existing channel

mask The edge detection mask.

edgeType The edge of the signal that triggers the state change detection event.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.615 UeiDaqAPI int UeiDaqSetDIIndustrialHighThreshold (ChannelHandle *channelHandle*, int *line*, double *highThreshold*)

The high input threshold is used in combination with the low input threshold to define an hysteresis window. The input line state will change only when the input signal crosses both threshold

Parameters:

channelHandle The handle to an existing industrial DI channel

line The input line to query or configure within the port

highThreshold the high hysteresis threshold

Returns:

The status code.

See also:

UeiDaqGetDIIndustrialHighThreshold (p. 806)

3.2.3.616 UeiDaqAPI int UeiDaqSetDIIndustrialInputGain (ChannelHandle *channelHandle*, int *line*, int *inputGain*)

Some DI devices use an ADC to measure the input line state. This setting provides a way to select the input gain for the ADC. The input gain value is an index in the gain list starting with 0.

Parameters:

channelHandle The handle to an existing industrial DI channel

line The input line to configure

inputGain the new input gain

Returns:

The status code.

3.2.3.617 UeiDaqAPI int UeiDaqSetDIIndustrialLowThreshold (ChannelHandle *channelHandle*, int *line*, double *lowThreshold*)

The low input threshold is used in combination with the high input threshold to define an hysteresis window. The input line state will change only when the input signal crosses both threshold

Parameters:

channelHandle The handle to an existing industrial DI channel

line The input line to query or configure within the port

lowThreshold the low hysteresis threshold

Returns:

The status code.

See also:

UeiDaqGetDIIndustrialLowThreshold (p. 807)

3.2.3.618 UeiDaqAPI int UeiDaqSetDIIndustrialMinimumPulseWidth (ChannelHandle *channelHandle*, int *line*, double *minWidth*)

The digital input filter is used to block digital noise from switching the input line state. It filters glitches or spikes based on their width.

Parameters:

channelHandle The handle to an existing industrial DI channel

line The input line to query or configure within the port

minWidth the minimum pulse width in ms. Use 0.0 to disable digital input filter.

Returns:

The status code.

See also:

UeiDaqGetDIIndustrialMinimumPulseWidth (p. 807)

3.2.3.619 UeiDaqAPI int UeiDaqSetDIIndustrialMux (ChannelHandle *channelHandle*, int *line*, tUeiDigitalInputMux *mux*)

Select how voltage supply is connected to each input line. Disconnected: Set the mux to tri-state mode to disconnect voltage supply from input line Diag mode: Set mux to Diagnostic mode to connect internal voltage source to each input line in order to test that it is functional Pull-up mode: Setting mux to pull-up mode connects a pull-up resistor between the internal voltage source and the input line to monitor switches or contacts without external circuitry.

Parameters:

channelHandle The handle to an existing industrial DI channel

line The input line to configure

mux the new mux state

Returns:

The status code.

3.2.3.620 UeiDaqAPI int UeiDaqSetDIIndustrialMuxDelay (ChannelHandle *channelHandle*, int *line*, int *muxDelayUs*)

Some DI devices use an ADC to measure the input line state. This setting provides a way to set the delay for the multiplexer in front of the ADC

Parameters:

channelHandle The handle to an existing industrial DI channel

line The input line to configure

muxDelayUs the mux delay in us

Returns:

The status code.

3.2.3.621 UeiDaqAPI int UeiDaqSetDMMFIRCutoff (ChannelHandle *channelHandle*, tUeiDMMFIRCutoff *frequency*)

Set the FIR cutoff frequency

Parameters:

channelHandle The handle to a DMM channel

frequency The new cutoff frequency setting

Returns:

The status code

3.2.3.622 UeiDaqAPI int UeiDaqSetDOChannelDefaultValue (ChannelHandle *channelHandle*, uInt32 *defaultVal*)

Set the default value.

Parameters:

channelHandle The handle to an existing channel

defaultVal the default value.

Returns:

The status code

See also:

UeiDaqGetChannelHandle (p. 778)

3.2.3.623 UeiDaqAPI int UeiDaqSetDOProtectedAutoRetry (ChannelHandle *channelHandle*, int *autoRetry*)

The auto retry setting specifies wheter the device should attempt to close the circuit after it was opened because of an over or under current condition. If it is set to true the device will try to close the circuit at a rate set by the autoRetryRate setting.

Parameters:

channelHandle The handle to an existing DO protected channel

autoRetry true to turn auto-retry on, false to turn it off.

Returns:

The status code.

See also:

UeiDaqGetDOProtectedAutoRetry (p. 809)

3.2.3.624 UeiDaqAPI int UeiDaqSetDOProtectedAutoRetryRate (ChannelHandle *channelHandle*, double *autoRetryRate*)

Specifies how often the device will attemp to close a circuit that was opened because of an over or under current condition.

Parameters:

channelHandle The handle to an existing DO protected channel
autoRetryRate The number of retries per second.

Returns:

The status code.

See also:

UeiDaqGetDOProtectedAutoRetryRate (p. 809)

3.2.3.625 UeiDaqAPI int UeiDaqSetDOProtectedCurrentMeasurementRate (ChannelHandle *channelHandle*, double *measurementRate*)

Set the rate at which the current is measured. This rate determines how fast the device react when an under or over current condition occurs.

Parameters:

channelHandle The handle to an existing DO protected channel
measurementRate The current measurement rate.

Returns:

The status code.

See also:

UeiDaqGetDOProtectedCurrentMeasurementRate (p. 810)

3.2.3.626 UeiDaqAPI int UeiDaqSetDOProtectedOverCurrentLimit (ChannelHandle *channelHandle*, int *line*, double *overCurrent*)

Set the maximum amount of current allowed in Amps. If more than the maximum current flows through the digital output line, the circuit will open.

Parameters:

channelHandle The handle to an existing DO protected channel
line The output line to configure within the port
overCurrent The over current limit.

Returns:

The status code.

See also:

UeiDaqGetDOProtectedUnderCurrentLimit (p. 812) **UeiDaqGetDOProtectedOverCurrentLimit** (p. 810) **UeiDaqSetDOProtectedUnderCurrentLimit** (p. 922)

3.2.3.627 UeiDaqAPI int UeiDaqSetDOProtectedOverUnderCount (ChannelHandle *channelHandle*, uInt32 *overUnderCount*)

Specifies number of consecutive over/under limit diagnostic readings that must occur in order to trip breaker.

Parameters:

channelHandle The handle to an existing channel

overUnderCount The new maximum number of over/under readings.

Returns:

The status code

3.2.3.628 UeiDaqAPI int UeiDaqSetDOProtectedPWMDutyCycle (ChannelHandle *channelHandle*, int *line*, double *dutyCycle*)

Specifies the continuous pulse train duty cycle

Parameters:

channelHandle The handle to an existing DO protected channel

line The output line to configure within the port

dutyCycle The new pulse train duty cycle.

Returns:

The status code.

See also:

UeiDaqGetDOProtectedPWMDutyCycle (p. 811)

3.2.3.629 UeiDaqAPI int UeiDaqSetDOProtectedPWMLength (ChannelHandle *channelHandle*, int *line*, uInt32 *lengthus*)

Specifies soft start/stop pulse train length in micro-seconds

Parameters:

channelHandle The handle to an existing DO protected channel

line The output line to configure within the port

lengthus The new pulse train length.

Returns:

The status code.

See also:

UeiDaqGetDOProtectedPWMLength (p. 811)

3.2.3.630 UeiDaqAPI int UeiDaqSetDOProtectedPWMMode (ChannelHandle *channelHandle*, int *line*, tUeiDOPWMMode *mode*)

Specifies the PWM mode for DO channels that support the capability With PWM mode you can replace the rising and falling edges with a pulse train to allow for soft start and/or stop

Parameters:

channelHandle The handle to an existing DO protected channel

line The output line to configure within the port

mode The new PWM mode.

Returns:

The status code.

See also:

UeiDaqGetDOProtectedPWMMode (p. 811)

3.2.3.631 UeiDaqAPI int UeiDaqSetDOProtectedPWMPeriod (ChannelHandle *channelHandle*, int *line*, uInt32 *periodus*)

Specifies soft start/stop or continuous pulse train period in micro-seconds

Parameters:

channelHandle The handle to an existing DO protected channel

line The output line to configure within the port

periodus The new pulse train period.

Returns:

The status code.

See also:

UeiDaqGetDOProtectedPWMPeriod (p. 812)

3.2.3.632 UeiDaqAPI int UeiDaqSetDOProtectedUnderCurrentLimit (ChannelHandle *channelHandle*, int *line*, double *underCurrent*)

Set the minimum amount of current allowed in Amps. If less than the minimum current flows through the digital output line, the circuit will open.

Parameters:

channelHandle The handle to an existing DO protected channel

line The output line to configure within the port

underCurrent The under current limit.

Returns:

The status code.

See also:

UeiDaqGetDOProtectedUnderCurrentLimit (p. 812) **UeiDaqGetDOProtectedOverCurrentLimit** (p. 810) **UeiDaqSetDOProtectedOverCurrentLimit** (p. 920)

3.2.3.633 UeiDaqAPI int UeiDaqSetI2CCustomSpeed (ChannelHandle *channelHandle*, float *bitRate*)

Set the I2C bus custom bit rate

Parameters:

channelHandle The handle to an I2C slave or master port

bitRate The new I2C bus custom bit rate

Returns:

The status code

3.2.3.634 UeiDaqAPI int UeiDaqSetI2CMasterByteToByteDelay (ChannelHandle *channelHandle*, uInt16 *delayUs*)

Set the delay in microseconds that the master will delay between sending bytes.

Parameters:

channelHandle The handle to an I2C master port

delayUs the new delay in microseconds

Returns:

The status code

3.2.3.635 UeiDaqAPI int UeiDaqSetI2CMasterLoopbackMode (ChannelHandle *channelHandle*, tUeiI2CLoopback *mode*)

Set the I2C loopback mode

Parameters:

channelHandle The handle to an I2C master port

mode The new I2C loopback mode

Returns:

The status code

3.2.3.636 UeiDaqAPI int UeiDaqSetI2CMasterMaxClockStretchingDelay (ChannelHandle *channelHandle*, uInt16 *delayUs*)

Set the delay in microseconds that the master will allow a slave to delay the clock

Parameters:

channelHandle The handle to an I2C master port

delayUs the new delay in microseconds

Returns:

The status code

3.2.3.637 UeiDaqAPI int UeiDaqSetI2CSlaveAddress (ChannelHandle *channelHandle*, int *address*)

Set the I2C slave address

Parameters:

channelHandle The handle to an I2C slave port

address The new I2C slave address

Returns:

The status code

3.2.3.638 UeiDaqAPI int UeiDaqSetI2CSlaveAddressWidth (ChannelHandle *channelHandle*, tUeiI2CSlaveAddressWidth *width*)

Set the I2C slave address width

Parameters:

channelHandle The handle to an I2C slave port

width The new I2C slave address width

Returns:

The status code

3.2.3.639 UeiDaqAPI int UeiDaqSetI2CSlaveClockStretchingDelay (ChannelHandle *channelHandle*, uInt16 *clocks*)

Get the delay in 15ns clock increments that the slave will delay an ACK. The clock stretching delay must also be individually enabled for address, transmit, and receive cycles. Max clock stretching delay is ~62ms.

Parameters:

channelHandle The handle to an I2C master port

clocks the delay in clocks

Returns:

The status code

3.2.3.640 UeiDaqAPI int UeiDaqSetI2CSlaveMaxWordsPerAck (ChannelHandle *channelHandle*, int *maxWords*)

Set maximum number of words slave will receive per transmission

Parameters:

channelHandle The handle to an I2C master port

maxWords the new maximum number of words slave will receive per transmission

Returns:

The status code

3.2.3.641 UeiDaqAPI int UeiDaqSetI2CSlaveRegisterData (ChannelHandle *channelHandle*, int *byteIndex*, uInt8 *data*)

Set the I2C slave padding data

Parameters:

channelHandle The handle to an I2C slave port

byteIndex Index of the byte to set, 0-3

data The new byte data

Returns:

The status code

3.2.3.642 UeiDaqAPI int UeiDaqSetI2CSlaveRegisterDataSize (ChannelHandle *channelHandle*, int *size*)

Set the number of bytes used in slave data when in Register data mode.

Parameters:

channelHandle The handle to an I2C master port

size the new number of bytes used in slave data when in Register data mode

Returns:

The status code

3.2.3.643 UeiDaqAPI int UeiDaqSetI2CSlaveTransmitDataMode (ChannelHandle *channelHandle*, tUeiI2CSlaveDataMode *dataMode*)

Set slave transmit data mode

Parameters:

channelHandle The handle to an I2C slave port

dataMode the new slave transmit data mode

Returns:

The status code

3.2.3.644 UeiDaqAPI int UeiDaqSetI2CSpeed (ChannelHandle *channelHandle*, tUeiI2CPortSpeed *bitRate*)

Set the I2C bus bit rate

Parameters:

channelHandle The handle to an I2C slave or master port

bitRate The new I2C bus bit rate

Returns:

The status code

3.2.3.645 UeiDaqAPI int UeiDaqSetI2CTTLLevel (ChannelHandle *channelHandle*, tUeiI2CTTLLevel *ttlLevel*)

Set the I2C bus TTL level

Parameters:

channelHandle The handle to an I2C slave or master port

ttlLevel The new I2C bus TTL level

Returns:

The status code

3.2.3.646 UeiDaqAPI int UeiDaqSetIRIGInputTimeCodeFormat (ChannelHandle *channelHandle*, tUeiIRIGTimeCodeFormat *format*)

Set the time code format used by the signal connected to the device.

Parameters:

channelHandle The handle to an existing IRIG channel

format the new time code format.

Returns:

The status code

See also:

UeiDaqGetIRIGTimeCodeFormat

3.2.3.647 UeiDaqAPI int UeiDaqSetIRIGInputTimeDecoderInput (ChannelHandle *channelHandle*, tUeiIRIGDecoderInputType *input*)

Set the type of time code/ connected to the device.

Parameters:

channelHandle The handle to an existing IRIG channel

input the new time decoder type.

Returns:

The status code

See also:

UeiDaqGetIRIGTimeDecoderInput

3.2.3.648 UeiDaqAPI int UeiDaqSetIRIGTimeKeeper1PPSSource (ChannelHandle *channelHandle*, tUeiIRIGTimeKeeper1PPSSource *source*)

For proper functioning, the timekeeper requires a 1PPS signal. which can be delivered from multiple sources: internal, external, GPS... This method sets the new time keeper 1PPS source

Parameters:

channelHandle The handle to an existing MIL-1553 channel

source the new 1 PPS source.

Returns:

The status code

See also:

UeiDaqGetIRIG1PPSSource

3.2.3.649 UeiDaqAPI int UeiDaqSetLVDTExcitationFrequency (ChannelHandle *channelHandle*, double *fex*)

Set the excitation frequency for this channel.

Parameters:

channelHandle The handle to an existing LVDT channel

fex The excitation frequency

Returns:

The status code.

See also:

UeiDaqGetLVDTExcitationFrequency (p. 817)

3.2.3.650 UeiDaqAPI int UeiDaqSetLVDTExcitationVoltage (ChannelHandle *channelHandle*, double *vex*)

Set the excitation RMS voltage for this channel.

Parameters:

channelHandle The handle to an existing LVDT channel

vex The excitation voltage

Returns:

The status code.

See also:

UeiDaqGetLVDTExcitationVoltage (p. 818)

3.2.3.651 UeiDaqAPI int UeiDaqSetLVDTSensorSensitivity (ChannelHandle *channelHandle*, double *sensitivity*)

Set the sensor sensitivity, it usually is in mVolts/g Independently of the unit, the voltage read will be converted to mV and divided by the excitation voltage and the sensitivity

Parameters:

channelHandle The handle to an existing LVDT channel

sensitivity The new sensitivity

Returns:

The status code.

See also:

UeiDaqGetLVDTSensorSensitivity (p. 818)

3.2.3.652 UeiDaqAPI int UeiDaqSetLVDTWiringScheme (ChannelHandle *channelHandle*, tUeiLVDTWiringScheme *wiring*)

Specifies the wiring scheme used to connect the resistive sensor to the channel.

Parameters:

channelHandle The handle to an existing LVDT channel

wiring The new wiring scheme.

Returns:

The status code.

See also:

UeiDaqGetLVDTWiringScheme (p. 818)

3.2.3.653 UeiDaqAPI int UeiDaqSetMIL1553Coupling (ChannelHandle *channelHandle*, tUeiMIL1553PortCoupling *coupling*)

Set the coupling needed to connect port to 1553 bus

Parameters:

channelHandle The handle to an existing MIL-1553 channel

coupling The 1553 port coupling

Returns:

The status code.

See also:

UeiDaqGetMIL1553Coupling (p. 819)

3.2.3.654 UeiDaqAPI int UeiDaqSetMIL1553PortMode (ChannelHandle *channelHandle*, tUeiMIL1553PortOpMode *portMode*)

Set the port mode of operation: BM, RT or BC

Parameters:

channelHandle The handle to an existing MIL-1553 channel

portMode The 1553 port mode of operation

Returns:

The status code.

See also:

UeiDaqGetMIL1553PortMode (p. 820)

3.2.3.655 UeiDaqAPI int UeiDaqSetMIL1553RxBus (ChannelHandle *channelHandle*, tUeiMIL1553PortActiveBus *portBus*)

Set the active bus: A or B or both

Parameters:

channelHandle The handle to an existing MIL-1553 channel

portBus The 1553 port active bus for reception

Returns:

The status code.

See also:

UeiDaqGetMIL1553RxBus (p. 820)

3.2.3.656 UeiDaqAPI int UeiDaqSetMIL1553TxBus (ChannelHandle *channelHandle*, tUeiMIL1553PortActiveBus *portBus*)

Set the active bus: A or B

Parameters:

channelHandle The handle to an existing MIL-1553 channel

portBus The 1553 port active bus for transmission

Returns:

The status code.

See also:

UeiDaqGetMIL1553TxBus (p. 821)

3.2.3.657 UeiDaqAPI int UeiDaqSetMuxOffDelay (ChannelHandle *channelHandle*, int *delay*)

Set the port off delay

Parameters:

channelHandle The handle to a MUX port

delay the new off delay

Returns:

The status code

3.2.3.658 UeiDaqAPI int UeiDaqSetMuxOnDelay (ChannelHandle *channelHandle*, int *delay*)

Set the port on delay

Parameters:

channelHandle The handle to a MUX port

delay the new on delay

Returns:

The status code

3.2.3.659 UeiDaqAPI int UeiDaqSetMuxSyncInputEdgePolarity (ChannelHandle *channelHandle*, tUeiDigitalEdge *polarity*)

Set the port synchronization input edge/level polarity

Parameters:

channelHandle The handle to a MUX port

polarity the new polarity

Returns:

The status code

3.2.3.660 UeiDaqAPI int UeiDaqSetMuxSyncOutputMode (ChannelHandle *channelHandle*, tUeiMuxSyncOutputMode *mode*)

Set the port synchronization output mode A MUX device can emit a pulse on its synchronization output when configuring its relays

Parameters:

channelHandle The handle to a MUX port

mode the new synchronization output mode

Returns:

The status code

3.2.3.661 UeiDaqAPI int UeiDaqSetMuxSyncOutputPulseWidth (ChannelHandle *channelHandle*, int *width*)

Set the port synchronization output oulse width in microseconds

Parameters:

channelHandle The handle to a MUX port

width the new synchronization output pulse width

Returns:

The status code

3.2.3.662 UeiDaqAPI int UeiDaqSetNumberOfPreTriggerScans (TriggerHandle *triggerHandle*, Int32 *numPreTriggerScans*)

The number of scans to save before the trigger occurs

Parameters:

triggerHandle The handle to an existing trigger object

numPreTriggerScans the new nunber of pre-trigger scans

Returns:

The status code

See also:

UeiDaqGetNumberOfPreTriggerScans (p. 823) **UeiDaqGetStartTriggerHandle** (p. 833) **UeiDaqGetStopTriggerHandle** (p. 833)

3.2.3.663 UeiDaqAPI int UeiDaqSetResistanceExcitationVoltage (ChannelHandle *channelHandle*, double *vex*)

Set the excitation voltage for this channel.

Parameters:

channelHandle The handle to an existing resistance channel

vex The excitation voltage

Returns:

The status code.

See also:

UeiDaqGetResistanceExcitationVoltage (p. 824)

3.2.3.664 UeiDaqAPI int UeiDaqSetResistanceReferenceResistance (ChannelHandle *channelHandle*, double *rref*)

Set the reference resistance used to measure the current flowing through the resistive sensor.

Parameters:

channelHandle The handle to an existing resistance channel

rref The reference resistance

Returns:

The status code.

See also:

UeiDaqGetResistanceReferenceResistance (p. 824)

3.2.3.665 UeiDaqAPI int UeiDaqSetResistanceReferenceResistorType (ChannelHandle *channelHandle*, tUeiReferenceResistorType *refResistorType*)

The reference resistor can be built-in the connector block if you are using a DNA-STP-AIU or connected externally.

Parameters:

channelHandle The handle to an existing resistance channel

refResistorType The reference resistor type.

Returns:

The status code.

See also:

UeiDaqGetResistanceReferenceResistorType (p. 824)

3.2.3.666 UeiDaqAPI int UeiDaqSetResistanceWiringScheme (ChannelHandle *channelHandle*, tUeiWiringScheme *wiring*)

Specifies the wiring scheme used to connect the resistive sensor to the channel.

Parameters:

channelHandle The handle to an existing resistance channel

wiring The new wiring scheme.

Returns:

The status code.

See also:

UeiDaqGetResistanceWiringScheme (p. 825)

3.2.3.667 UeiDaqAPI int UeiDaqSetRTDCoefficientA (ChannelHandle *channelHandle*, double *A*)

The Callendar-Van Dusen equation is an equation that describes the relationship between resistance (R) and temperature (t) of platinum resistance thermometers. t < 0: R(t) = R(0)[1 + A ∗ t + B ∗ t ∗ t + (t - 100)C ∗ t ∗ t ∗ t]. t > 0: R(t) = R(0)(1 + A ∗ t + B ∗ t ∗ t).

Parameters:

channelHandle The handle to an existing resistance channel

A The CVD A coefficient

Returns:

The status code.

See also:

UeiDaqGetRTDCoefficientA (p. 825)

3.2.3.668 UeiDaqAPI int UeiDaqSetRTDCoefficientB (ChannelHandle *channelHandle*, double *B*)

The Callendar-Van Dusen equation is an equation that describes the relationship between resistance (R) and temperature (t) of platinum resistance thermometers. t < 0: R(t) = R(0)[1 + A ∗ t + B ∗ t ∗ t + (t - 100)C ∗ t ∗ t ∗ t]. t > 0: R(t) = R(0)(1 + A ∗ t + B ∗ t ∗ t).

Parameters:

channelHandle The handle to an existing resistance channel

B The CVD B coefficient

Returns:

The status code.

See also:

UeiDaqGetRTDCoefficientB (p. 825)

3.2.3.669 UeiDaqAPI int UeiDaqSetRTDCoefficientC (ChannelHandle *channelHandle*, double *C*)

The Callendar-Van Dusen equation is an equation that describes the relationship between resistance (R) and temperature (t) of platinum resistance thermometers. t < 0: R(t) = R(0)[1 + A ∗ t + B ∗ t ∗ t + (t - 100)C ∗ t ∗ t ∗ t]. t > 0: R(t) = R(0)(1 + A ∗ t + B ∗ t ∗ t).

Parameters:

channelHandle The handle to an existing resistance channel

C The CVD C coefficient

Returns:

The status code.

See also:

UeiDaqGetRTDCoefficientC (p. 826)

3.2.3.670 UeiDaqAPI int UeiDaqSetRTDNominalResistance (ChannelHandle *channelHandle*, double *r0*)

Set the RTD nominal resistance at 0 deg.

Parameters:

channelHandle The handle to an existing resistance channel

r0 The RTD nominal resistance

Returns:

The status code.

See also:

UeiDaqGetRTDNominalResistance (p. 826)

3.2.3.671 UeiDaqAPI int UeiDaqSetRTDTemperatureScale (ChannelHandle *channelHandle*, tUeiTemperatureScale *tempScale*)

Set the temperature scale used to convert the measured resistance to temperature.

Parameters:

channelHandle The handle to an existing RTD channel

tempScale the temperature scale.

Returns:

The status code.

See also:

UeiDaqGetTCTemperatureScale (p. 836)

3.2.3.672 UeiDaqAPI int UeiDaqSetRTDType (ChannelHandle *channelHandle*, tUeiRTDType *type*)

RTD sensors are specified using the "alpha" (a) constant It is also known as the temperature coefficient of resistance, and symbolizes the resistance change factor per degree of temperature change. The RTD type is used to select the proper coefficients A, B and C for the Callendar Van-Dusen equation used to convert resistance measurements to temperature.

Parameters:

channelHandle The handle to an existing resistance channel

type The RTD type

Returns:

The status code.

See also:

UeiDaqGetRTDType (p. 827)

3.2.3.673 UeiDaqAPI int UeiDaqSetSerialPortCustomSpeed (ChannelHandle *channelHandle*, uInt32 *bitsPerSecond*)

Set the number of data bits transmitted per second. Call this function when using a non-standard port speed

Parameters:

channelHandle The handle to an existing serial port

bitsPerSecond The serial port speed

Returns:

The status code.

See also:

UeiDaqGetSerialPortCustomSpeed (p. 827)

3.2.3.674 UeiDaqAPI int UeiDaqSetSerialPortDataBits (ChannelHandle *channelHandle*, tUeiSerialPortDataBits *dataBits*)

Set the number of data bits that hold the data to be transferred.

Parameters:

channelHandle The handle to an existing serial port

dataBits The number of data bits

Returns:

The status code.

See also:

UeiDaqGetSerialPortDataBits (p. 827)

3.2.3.675 UeiDaqAPI int UeiDaqSetSerialPortFlowControl (ChannelHandle *channelHandle*, tUeiSerialPortFlowControl *flowControl*)

Set flow control setting

Parameters:

channelHandle The handle to an existing serial port

flowControl The new flow control setting

Returns:

The status code.

See also:

UeiDaqGetSerialPortFlowControl (p. 828)

3.2.3.676 UeiDaqAPI int UeiDaqSetSerialPortMode (ChannelHandle *channelHandle*, tUeiSerialPortMode *mode*)

Set the mode used by the serial port. Possible modes are RS-232, RS-485 half duplex and RS-485 full duplex

Parameters:

channelHandle The handle to an existing serial port

mode The new serial port mode

Returns:

The status code.

See also:

UeiDaqGetSerialPortMode (p. 828)

3.2.3.677 UeiDaqAPI int UeiDaqSetSerialPortParity (ChannelHandle *channelHandle*, tUeiSerialPortParity *parity*)

Set the parity used to detect transmission errors.

Parameters:

channelHandle The handle to an existing serial port

parity The serial port parity

Returns:

The status code.

See also:

UeiDaqGetSerialPortParity (p. 828)

3.2.3.678 UeiDaqAPI int UeiDaqSetSerialPortSpeed (ChannelHandle *channelHandle*, tUeiSerialPortSpeed *bitsPerSecond*)

Set the number of data bits transmitted per second.

Parameters:

channelHandle The handle to an existing serial port

bitsPerSecond The serial port speed

Returns:

The status code.

See also:

UeiDaqGetSerialPortSpeed (p. 829)

3.2.3.679 UeiDaqAPI int UeiDaqSetSerialPortStopBits (ChannelHandle *channelHandle*, tUeiSerialPortStopBits *stopBits*)

Set the number of stop bits that indicate the end of a data message.

Parameters:

channelHandle The handle to an existing serial port

stopBits The number of stop bits

Returns:

The status code.

See also:

UeiDaqGetSerialPortStopBits

3.2.3.680 UeiDaqAPI int UeiDaqSetSerialPortTermination (ChannelHandle *channelHandle*, char * *eol*)

Set the sequence of characters used to define the end of a line.

Parameters:

channelHandle The handle to an existing serial port

eol The termination characters

Returns:

The status code.

See also:

UeiDaqGetSerialPortTermination (p. 829)

3.2.3.681 UeiDaqAPI int UeiDaqSetSessionCustomProperty (SessionHandle *sessionHandle*, char * *property*, int *valueSize*, void * *value*)

Set a custom property on the device or subsystem associated with this session Custom properties are device dependent, refer to the device user manual to learn about the properties supported by your device

Parameters:

sessionHandle The handle of the session

property string containing the name of the property to set

valueSize size in bytes of the memory block containing the property value

value pointer to the memory block containing the property value

Returns:

The status code

3.2.3.682 UeiDaqAPI int UeiDaqSetSetTriggerChannel (TriggerHandle *triggerHandle*, Int32 *channel*)

Set the channel to monitor for the trigger condition.

Parameters:

triggerHandle The handle to an existing trigger object

channel the new channel to monitor

Returns:

The status code

See also:

UeiDaqGetTriggerChannel (p. 843) **UeiDaqGetStartTriggerHandle** (p. 833) **UeiDaqGetStopTriggerHandle** (p. 833)

3.2.3.683 UeiDaqAPI int UeiDaqSetSimulatedLVDTExcitationFrequency (ChannelHandle *channelHandle*, double *fex*)

Set the excitation frequency for this channel.

Parameters:

channelHandle The handle to an existing simulated LVDT channel

fex The excitation frequency

Returns:

The status code.

See also:

UeiDaqGetSimulatedLVDTExcitationFrequency (p. 830)

3.2.3.684 UeiDaqAPI int UeiDaqSetSimulatedLVDTExcitationVoltage (ChannelHandle *channelHandle*, double *vex*)

Set the excitation RMS voltage for this channel.

Parameters:

channelHandle The handle to an existing simulated LVDT channel

vex The excitation voltage

Returns:

The status code.

See also:

UeiDaqGetSimulatedLVDTExcitationVoltage (p. 831)

3.2.3.685 UeiDaqAPI int UeiDaqSetSimulatedLVDTSensorSensitivity (ChannelHandle *channelHandle*, double *sensitivity*)

Set the sensor sensitivity, it usually is in mVolts/g Independently of the unit, the voltage read will be converted to mV and divided by the excitation voltage and the sensitivity

Parameters:

channelHandle The handle to an existing simulated LVDT channel

sensitivity The new sensitivity

Returns:

The status code.

See also:

UeiDaqGetSimulatedLVDTSensorSensitivity (p. 831)

3.2.3.686 UeiDaqAPI int UeiDaqSetSimulatedLVDTWiringScheme (ChannelHandle *channelHandle*, tUeiLVDTWiringScheme *wiring*)

Specifies the wiring scheme used to connect the resistive sensor to the channel.

Parameters:

channelHandle The handle to an existing simulated LVDT channel

wiring The new wiring scheme.

Returns:

The status code.

See also:

UeiDaqGetSimulatedLVDTWiringScheme (p. 831)

3.2.3.687 UeiDaqAPI int UeiDaqSetSimulatedSynchroResolverExcitationFrequency (ChannelHandle *channelHandle*, double *fex*)

Set the excitation frequency for this channel.

Parameters:

channelHandle The handle to an existing simulated synchro/resolver channel

fex The excitation frequency

Returns:

The status code.

See also:

UeiDaqGetSimulatedSynchroResolverExcitationFrequency (p. 832)

3.2.3.688 UeiDaqAPI int UeiDaqSetSimulatedSynchroResolverExcitationVoltage (ChannelHandle *channelHandle*, double *vex*)

Set the excitation RMS voltage for this channel.

Parameters:

channelHandle The handle to an existing simulated synchro/resolver channel

vex The excitation voltage

Returns:

The status code.

See also:

UeiDaqGetSimulatedSynchroResolverExcitationVoltage (p. 832)

3.2.3.689 UeiDaqAPI int UeiDaqSetSimulatedSynchroResolverMode (ChannelHandle *channelHandle*, tUeiSynchroResolverMode *mode*)

Specifies whether the sensor simulated is a synchro or a resolver.

Parameters:

channelHandle The handle to an existing simulated synchro/resolver channel

mode The new simulated sensor type.

Returns:

The status code.

See also:

UeiDaqGetSimulatedSynchroResolverMode (p. 832)

3.2.3.690 UeiDaqAPI int UeiDaqSetSynchroResolverExcitationFrequency (ChannelHandle *channelHandle*, double *fex*)

Set the excitation frequency for this channel.

Parameters:

channelHandle The handle to an existing synchro/resolver channel

fex The excitation frequency

Returns:

The status code.

See also:

UeiDaqGetSynchroResolverExcitationFrequency (p. 833)

3.2.3.691 UeiDaqAPI int UeiDaqSetSynchroResolverExcitationVoltage (ChannelHandle *channelHandle*, double *vex*)

Set the excitation RMS voltage for this channel.

Parameters:

channelHandle The handle to an existing synchro/resolver channel

vex The excitation voltage

Returns:

The status code.

See also:

UeiDaqGetSynchroResolverExcitationVoltage (p. 834)

3.2.3.692 UeiDaqAPI int UeiDaqSetSynchroResolverMode (ChannelHandle *channelHandle*, tUeiSynchroResolverMode *mode*)

Specifies whether the input sensor is a synchro or a resolver.

Parameters:

channelHandle The handle to an existing synchro/resolver channel

mode The new input sensor type.

Returns:

The status code.

See also:

UeiDaqGetSynchroResolverMode (p. 834)

3.2.3.693 UeiDaqAPI int UeiDaqSetTCCJCConstant (ChannelHandle *channelHandle*, double *cjcConstant*)

Set the cold junction compensation temperature constant. This setting is only used when the CJC type is set to UeiCJCTypeConstant. The unit is the same as the configured temperature scale.

Parameters:

channelHandle The handle to an existing TC channel

cjcConstant the cold junction compensation constant.

Returns:

The status code.

See also:

UeiDaqGetTCCJCConstant (p. 835)

3.2.3.694 UeiDaqAPI int UeiDaqSetTCCJCResource (ChannelHandle *channelHandle*, char * *cjcResource*)

Set the resource string describing how to measure the cold junction temperature. This setting is only used when the CJC type is set to UeiCJCTypeResource. The CJC resource format is similar to the channel list format: [cjcSensorType]://[Channel(s)]?[Paramters] For example: to measure CJC from a linear IC sensor connected to channel 20 the resource string is "ic://20?K=0.00295" this will measure the voltage on channel 20 and compute the CJC temperature with the formula: degK = V / 0.00295

Parameters:

channelHandle The handle to an existing TC channel

cjcResource the cold junction compensation resource.

Returns:

The status code.

See also:

UeiDaqGetTCCJCResource (p. 835)

3.2.3.695 UeiDaqAPI int UeiDaqSetTCCJCType (ChannelHandle *channelHandle*, tUeiColdJunctionCompensationType *cjcType*)

Set the type of the cold junction compensation used to convert from volts to temperature.

Parameters:

channelHandle The handle to an existing TC channel

cjcType the cold junction compensation type.

Returns:

The status code.

See also:

UeiDaqGetTCCJCType (p. 835)

3.2.3.696 UeiDaqAPI int UeiDaqSetTCTemperatureScale (ChannelHandle *channelHandle*, tUeiTemperatureScale *tempScale*)

Set the temperature scale used to convert the measured voltage to temperature.

Parameters:

channelHandle The handle to an existing TC channel

tempScale the temperature scale.

Returns:

The status code.

See also:

UeiDaqGetTCTemperatureScale (p. 836)

3.2.3.697 UeiDaqAPI int UeiDaqSetTCThermocoupleType (ChannelHandle *channelHandle*, tUeiThermocoupleType *tcType*)

Set the type of the thermocouple connected to the channel.

Parameters:

channelHandle The handle to an existing TC channel
tcType the thermocouple type.

Returns:

The status code.

See also:

UeiDaqGetTCThermocoupleType (p. 836)

3.2.3.698 UeiDaqAPI int UeiDaqSetTimingConvertClockDestinationSignal (TimingHandle *timingHandle*, char * *signal*)

Set the name of the signal driven by the conversion clock. The signal name is device dependent.

Parameters:

timingHandle The handle to an existing timing object
signal the new conversion clock destination signal

Returns:

the status code

See also:

UeiDaqGetTimingConvertClockDestinationSignal (p. 836)

3.2.3.699 UeiDaqAPI int UeiDaqSetTimingConvertClockEdge (TimingHandle *timingHandle*, tUeiDigitalEdge *edge*)

Set the active edge of the conversion clock.

Parameters:

timingHandle The handle to an existing timing object
edge The new active edge

Returns:

The status code

See also:

UeiDaqGetTimingHandle (p. 839)

3.2.3.700 UeiDaqAPI int UeiDaqSetTimingConvertClockRate (TimingHandle *timingHandle*, f64 *rate*)

Set the rate of the conversion clock

Parameters:

timingHandle The handle to an existing timing object

rate The new clock rate

Returns:

The status code

See also:

UeiDaqGetTimingHandle (p. 839)

3.2.3.701 UeiDaqAPI int UeiDaqSetTimingConvertClockSource (TimingHandle *timingHandle*, tUeiTimingClockSource *source*)

Set the source of the A/D conversion clock

Parameters:

timingHandle The handle to an existing timing object

source The new clock source

Returns:

The status code

See also:

UeiDaqGetTimingHandle (p. 839)

3.2.3.702 UeiDaqAPI int UeiDaqSetTimingConvertClockSourceSignal (TimingHandle *timingHandle*, char * *signal*)

Set the name of the signal to use as conversion clock. The signal name is device dependent.

Parameters:

timingHandle The handle to an existing timing object

signal the new conversion clock signal

Returns:

the status code

See also:

UeiDaqGetTimingConvertClockSourceSignal (p. 838)

3.2.3.703 UeiDaqAPI int UeiDaqSetTimingConvertClockTimebaseDividingCounter (TimingHandle *timingHandle*, int *counter*)

Set the convert clock dividing counter. This counter is used to divide an external.

Parameters:

timingHandle The handle to an existing timing object

counter the timebase dividing counter

Returns:

the status code

See also:

UeiDaqTimingGetConvertClockTimebaseDividingCounter (p. 956)

3.2.3.704 UeiDaqAPI int UeiDaqSetTimingConvertClockTimebaseDivisor (TimingHandle *timingHandle*, int *divisor*)

Set the convert clock divisor. This divisor is used to divide an external clock using one of the on-board counter/timers.

Parameters:

timingHandle The handle to an existing timing object

divisor the timebase divisor

Returns:

the status code

See also:

UeiDaqGetTimingConvertClockTimebaseDivisor (p. 838)

3.2.3.705 UeiDaqAPI int UeiDaqSetTimingDuration (TimingHandle *timingHandle*, tUeiTimingDuration *duration*)

Set the duration of the timed operation.

Parameters:

timingHandle The handle to an existing timing object

duration The new duration

Returns:

The status code

See also:

UeiDaqGetTimingHandle (p. 839)

3.2.3.706 UeiDaqAPI int UeiDaqSetTimingMode (TimingHandle *timingHandle*, tUeiTimingMode *mode*)

Set the timing mode

Parameters:

timingHandle The handle to an existing timing object

mode The new timing mode

Returns:

The status code

See also:

UeiDaqGetTimingHandle (p. 839)

3.2.3.707 UeiDaqAPI int UeiDaqSetTimingScanClockDestinationSignal (TimingHandle *timingHandle*, char * *signal*)

Set the name of the signal to use as Scan clock. The signal name is device dependent.

Parameters:

timingHandle The handle to an existing timing object

signal the new scan clock destination signal

Returns:

the status code

See also:

UeiDaqGetTimingScanClockDestinationSignal (p. 840)

3.2.3.708 UeiDaqAPI int UeiDaqSetTimingScanClockEdge (TimingHandle *timingHandle*, tUeiDigitalEdge *edge*)

Set the active edge of the scan clock.

Parameters:

timingHandle The handle to an existing timing object

edge The new active edge

Returns:

The status code

See also:

UeiDaqGetTimingHandle (p. 839)

3.2.3.709 UeiDaqAPI int UeiDaqSetTimingScanClockRate (TimingHandle *timingHandle*, f64 *rate*)

Set the rate of the scan clock

Parameters:

timingHandle The handle to an existing timing object

rate The new clock rate

Returns:

The status code

See also:

UeiDaqGetTimingHandle (p. 839)

3.2.3.710 UeiDaqAPI int UeiDaqSetTimingScanClockSource (TimingHandle *timingHandle*, tUeiTimingClockSource *source*)

Set the source of the scan clock

Parameters:

timingHandle The handle to an existing timing object

source The new clock source

Returns:

The status code

See also:

UeiDaqGetTimingHandle (p. 839)

3.2.3.711 UeiDaqAPI int UeiDaqSetTimingScanClockSourceSignal (TimingHandle *timingHandle*, char * *signal*)

Set the name of the signal to use as Scan clock. The signal name is device dependent.

Parameters:

timingHandle The handle to an existing timing object

signal the new scan clock source signal

Returns:

the status code

See also:

UeiDaqGetTimingScanClockSourceSignal (p. 841)

3.2.3.712 UeiDaqAPI int UeiDaqSetTimingScanClockTimebaseDividingCounter (TimingHandle *timingHandle*, int *counter*)

Set the scan clock dividing counter. This counter is used to divide an external clock.

Parameters:

timingHandle The handle to an existing timing object

counter the timebase dividing counter

Returns:

the status code

See also:

UeiDaqGetTimingScanClockTimebaseDividingCounter (p. 841)

3.2.3.713 UeiDaqAPI int UeiDaqSetTimingScanClockTimebaseDivisor (TimingHandle *timingHandle*, int *divisor*)

Set the scan clock divisor. This divisor is used to divide an external clock using one of the on-board counter/timers.

Parameters:

timingHandle The handle to an existing timing object

divisor the timebase divisor

Returns:

the status code

See also:

UeiDaqGetTimingScanClockTimebaseDivisor (p. 842)

3.2.3.714 UeiDaqAPI int UeiDaqSetTimingTimeout (TimingHandle *timingHandle*, Int32 *timeout*)

Set the delay after which the session is aborted.

Parameters:

timingHandle The handle to an existing timing object

timeout The new timeout in ms

Returns:

The status code

See also:

UeiDaqGetTimingHandle (p. 839)

3.2.3.715 UeiDaqAPI int UeiDaqSetTriggerAction (TriggerHandle *triggerHandle*, tUeiTriggerAction *action*)

Set the action to execute when the trigger will occur

Parameters:

triggerHandle The handle to an existing trigger object

action the new trigger action

Returns:

The status code

See also:

UeiDaqGetTriggerAction (p. 842) **UeiDaqGetStartTriggerHandle** (p. 833) **UeiDaqGetStopTriggerHandle** (p. 833)

3.2.3.716 UeiDaqAPI int UeiDaqSetTriggerCondition (TriggerHandle *triggerHandle*, tUeiTriggerCondition *condition*)

Set the condition for the analog software trigger

Parameters:

triggerHandle The handle to an existing trigger object

condition the new condition

Returns:

The status code

See also:

UeiDaqGetTriggerCondition (p. 843) **UeiDaqGetStartTriggerHandle** (p. 833) **UeiDaqGetStopTriggerHandle** (p. 833)

3.2.3.717 UeiDaqAPI int UeiDaqSetTriggerDestinationSignal (TriggerHandle *triggerHandle*, char * *signal*)

Set the signal where the trigger is routed out of the device. The signal name is device dependent.

Parameters:

triggerHandle The handle to an existing trigger object

signal the new trigger destination signal

Returns:

The status code

See also:

UeiDaqGetTriggerDestinationSignal (p. 843)

3.2.3.718 UeiDaqAPI int UeiDaqSetTriggerDigitalEdge (TriggerHandle *triggerHandle*, tUeiDigitalEdge *edge*)

Set the active edge of the trigger signal

Parameters:

triggerHandle The handle to an existing trigger object

edge The new trigger edge

Returns:

The status code

See also:

UeiDaqGetStartTriggerHandle (p. 833) **UeiDaqGetStopTriggerHandle** (p. 833)

3.2.3.719 UeiDaqAPI int UeiDaqSetTriggerHysteresis (TriggerHandle *triggerHandle*, f64 *hysteresis*)

Set the hysteresis window scaled value. The hysteresis window is a noise filter. The trigger occurs when the signal leave the window.

Parameters:

triggerHandle The handle to an existing trigger object

hysteresis the new hysteresis

Returns:

The status code

See also:

UeiDaqGetTriggerHysteresis (p. 844) **UeiDaqGetStartTriggerHandle** (p. 833) **UeiDaqGetStopTriggerHandle** (p. 833)

3.2.3.720 UeiDaqAPI int UeiDaqSetTriggerLevel (TriggerHandle *triggerHandle*, f64 *level*)

Set the scaled value at which the trigger occurs. The value must be specified in the same unit as the measurement.

Parameters:

triggerHandle The handle to an existing trigger object

level the new level

Returns:

The status code

See also:

UeiDaqGetTriggerLevel (p. 844) **UeiDaqGetStartTriggerHandle** (p. 833) **UeiDaqGetStopTriggerHandle** (p. 833)

3.2.3.721 UeiDaqAPI int UeiDaqSetTriggerSource (TriggerHandle *triggerHandle*, tUeiTriggerSource *source*)

Set the source of the signal that will trigger the start of the session

Parameters:

triggerHandle The handle to an existing trigger object

source The new trigger source

Returns:

The status code

See also:

UeiDaqGetStartTriggerHandle (p. 833) **UeiDaqGetStopTriggerHandle** (p. 833)

3.2.3.722 UeiDaqAPI int UeiDaqSetTriggerSourceSignal (TriggerHandle *triggerHandle*, char * *signal*)

Set the signal used to transmit the trigger to the device. The signal name is device dependent.

Parameters:

triggerHandle The handle to an existing trigger object

signal the new trigger source signal

Returns:

The status code

See also:

UeiDaqGetTriggerSourceSignal (p. 845)

3.2.3.723 UeiDaqAPI int UeiDaqSetVRADCMovingAverage (ChannelHandle *channelHandle*, int *mvAvrg*)

Set the size of the moving average window applied to the VR sensor signal while it is measured.

Parameters:

channelHandle The handle to an existing VR channel

mvAvrg The new ADC moving average.

Returns:

The status code

3.2.3.724 UeiDaqAPI int UeiDaqSetVRADCRate (ChannelHandle *channelHandle*, double *rate*)

Set the rate at which the VR sensor is measured.

Parameters:

channelHandle The handle to an existing VR channel

rate The new ADC rate.

Returns:

The status code

3.2.3.725 UeiDaqAPI int UeiDaqSetVRAPTMode (ChannelHandle *channelHandle*, tUeiVRAPTMode *mode*)

APT finds the point in time where the VR sensor output voltage falls below a certain threshold. This point marks the beginning of the gap between two teeth.

Parameters:

channelHandle The handle to an existing VR channel

mode the new APT mode

Returns:

The status code

3.2.3.726 UeiDaqAPI int UeiDaqSetVRAPTThreshold (ChannelHandle *channelHandle*, double *threshold*)

The APT threshold is used when APT mode is set to "Fixed"

Parameters:

channelHandle The handle to an existing VR channel

threshold The new APT threshold.

Returns:

The status code

3.2.3.727 UeiDaqAPI int UeiDaqSetVRAPTThresholdDivider (ChannelHandle *channelHandle*, int *divider*)

The APT threshold divider is used when APT mode is set to "Logic"

Parameters:

channelHandle The handle to an existing VR channel

divider The new APT threshold divider.

Returns:

The status code

3.2.3.728 UeiDaqAPI int UeiDaqSetVRMode (ChannelHandle *channelHandle*, tUeiVRMode *mode*)

The VR-608 can use four different modes to measure velocity, position or direction The counting mode can be set to: Decoder: Even and Odd channels are used in pair to determine direction and position Timed: Count number of teeth detected during a timed interval N pulses: Measure the time taken to detect N teeth (Number of teeth needs to be set separately) Z pulse: Measure the number of teeth and the time elapsed between two Z pulses (The Z tooth is usually a gap or a double tooth on the encoder wheel)

In decoder mode the settings for the even channel are applied to both even and odd channels.

Parameters:

channelHandle The handle to an existing VR channel

mode the new VR mode

Returns:

The status code

3.2.3.729 UeiDaqAPI int UeiDaqSetVRNumberOfTeeth (ChannelHandle *channelHandle*, int *numTeeth*)

The number of teeth on the encoder wheel

Parameters:

channelHandle The handle to an existing VR channel

numTeeth The new number of teeth.

Returns:

The status code

3.2.3.730 UeiDaqAPI int UeiDaqSetVRZCLevel (ChannelHandle *channelHandle*, double *level*)

The ZC level is used when ZC mode is set to "Fixed"

Parameters:

channelHandle The handle to an existing VR channel

level The new ZC level.

Returns:

The status code

3.2.3.731 UeiDaqAPI int UeiDaqSetVRZCMode (ChannelHandle *channelHandle*, tUeiVRZCMode *mode*)

Zero crossing finds the point in time where the VR sensor output voltage goes from positive to negative voltage. This point is when the center of the tooth is lining up with the center of the VR sensor.

Parameters:

channelHandle The handle to an existing VR channel

mode the new zero-crossing mode

Returns:

The status code

3.2.3.732 UeiDaqAPI int UeiDaqStartSession (SessionHandle *sessionHandle*)

Start the session immediately of after the trigger condition occurred.

Parameters:

sessionHandle The handle to an existing session

Returns:

The status code

See also:

UeiDaqStopSession (p. 955) **UeiDaqCleanUpSession** (p. 707) **UeiDaqIsSessionRunning** (p. 865) **UeiDaqWaitUntilSessionIsDone** (p. 957)

3.2.3.733 UeiDaqAPI int UeiDaqStopSession (SessionHandle *sessionHandle*)

Stop the session immediately and wait for the session to be in the stopped state before returning, once it returned you can restart the session immediately with **UeiDaqStartSession()** (p. 955) without having to reconfigure channels. timing or triggers.

Parameters:

sessionHandle The handle to an existing session

Returns:

The status code

See also:

UeiDaqStartSession (p. 955) **UeiDaqCleanUpSession** (p. 707) **UeiDaqIsSessionRunning** (p. 865) **UeiDaqWaitUntilSessionIsDone** (p. 957)

3.2.3.734 UeiDaqAPI int UeiDaqTimingGetConvertClockTimebaseDividingCounter (TimingHandle *timingHandle*, int * *counter*)

Get the convert clock dividing counter. This counter is used to divide an external clock.

Parameters:

timingHandle The handle to an existing timing object

counter the timebase dividing counter

Returns:

the status code

See also:

UeiDaqSetTimingConvertClockTimebaseDividingCounter (p. 946)

3.2.3.735 UeiDaqAPI const char* UeiDaqTranslateError (int *error*)

Translate error code to a human readable message

Parameters:

error The error code to translate

Returns:

The error message

3.2.3.736 UeiDaqAPI int UeiDaqTriggerSync1PPSDevices (SessionHandle *sessionHandle*, tUeiSync1PPSTriggerOperation *trigOp*, int *resetTimestamp*)

Set and send the trigger signal that will start all slave deivces

Parameters:

sessionHandle The handle to an existing session

trigOp The trigger operation

resetTimestamp true to reset timestamp counter on all slave devices

Returns:

The status code

3.2.3.737 UeiDaqAPI int UeiDaqUpdateSynchroResolverAngles (SessionHandle *sessionHandle*, Int32 * *written*, Int32 * *available*)

Update all channels with staged angles and delays

Parameters:

sessionHandle The handle to an existing session

written array containing the number of values actually written for each channel

available array containing the space available in FIFO for each channel

Returns:

The status code

3.2.3.738 UeiDaqAPI int UeiDaqWaitUntilSessionIsDone (SessionHandle *sessionHandle*, int *timeout*, int * *done*)

Wait for the specified time until the session is stopped.

Parameters:

sessionHandle The handle to an existing session

timeout Maximum amount of time in ms this call will wait.

done True if the session is finished, false otherwise

Returns:

The status code

See also:

UeiDaqStartSession (p. 955) **UeiDaqStopSession** (p. 955) **UeiDaqCleanUpSession** (p. 707) **UeiDaqIsSessionRunning** (p. 865)

3.2.3.739 UeiDaqAPI int UeiDaqWriteAOWaveform (SessionHandle *sessionHandle*, Int32 *channel*, Int32 *numWfms*, tUeiAOWaveformParameters * *pBuffer*, Int32 * *numWfmsWritten*)

Format and send one or more tUeiAOWaveformParameters(s) to the AO device.

Parameters:

sessionHandle The handle to an existing session

channel the channel to write the waveform to

numWfms the number of waveforms to send

pBuffer source buffer

numWfmsWritten the actual number of waveforms sent

Returns:

The status code

3.2.3.740 UeiDaqAPI int UeiDaqWriteAOWaveformArbitraryData (SessionHandle *sessionHandle*, Int32 *channel*, Int32 *numValues*, double * *pBuffer*, Int32 * *numValsWritten*)

Send arbitrary waveform data to the AO device.

Parameters:

sessionHandle The handle to an existing session
channel the channel to write the waveform to
numValues the number of values to send
pBuffer source buffer
numValsWritten the actual number of values sent

Returns:

The status code

3.2.3.741 UeiDaqAPI int UeiDaqWriteAOWaveformSweep (SessionHandle *sessionHandle*, Int32 *channel*, Int32 *numSwps*, tUeiAOWaveformSweepParameters * *pBuffer*, Int32 * *numSwpsWritten*)

Format and send one or more tUeiAOWaveformSweepParameter(s) to the AO device.

Parameters:

sessionHandle The handle to an existing session
channel the channel to sweep
numSwps the number of sweep commands to send
pBuffer source buffer
numSwpsWritten the actual number of sweep commands sent

Returns:

The status code

3.2.3.742 UeiDaqAPI int UeiDaqWriteARINCRawWords (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numWords*, uInt32 * *pBuffer*, Int32 * *numWordsWritten*)

Write the requested number of raw ARINC words to the port. Raw ARINC word is a 32 bits value encoding the ARINC word fields: SSM, SDI, Label, Data and Parity

Parameters:

sessionHandle The handle to an existing session
port the port to write the message to
numWords Number of words to write
pBuffer Source buffer for the words
numWordsWritten Number of words actually written to the port

Returns:

The status code

3.2.3.743 UeiDaqAPI int UeiDaqWriteARINCWords (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numWords*, tUeiARINCWord * *pBuffer*, Int32 * *numWordsWritten*)

Write the requested number of ARINC words to the port

Parameters:

sessionHandle The handle to an existing session
port the port to write the message to
numWords Number of words to write
pBuffer Source buffer for the words
numWordsWritten Number of words actually written to the port

Returns:

The status code

3.2.3.744 UeiDaqAPI int UeiDaqWriteARINCWordsAsync (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numWords*, tUeiARINCWord * *pBuffer*, tUeiEventCallback *pEventCallback*)

Write the requested number of ARINC words asynchronously to the port This function returns immediately, the specified callback function is called once the buffer has been consumed. The callback function parameter contains the number of words actually written.

Parameters:

sessionHandle The handle to an existing session
port the port to write the message to
numWords Number of words to write
pBuffer Source buffer for the words
pEventCallback The callback function pointer

Returns:

The status code

3.2.3.745 UeiDaqAPI int UeiDaqWriteCANFrame (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numFrames*, tUeiCANFrame * *pBuffer*, Int32 * *numFramesWritten*)

Write the requested number of CAN frames to the port

Parameters:

sessionHandle The handle to an existing session
port the port to write the message to
numFrames Number of frames to write
pBuffer Source buffer for the frames
numFramesWritten Number of frames actually written to the port

Returns:

The status code

3.2.3.746 UeiDaqAPI int UeiDaqWriteCANFrameAsync (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numFrames*, tUeiCANFrame * *pBuffer*, tUeiEventCallback *pEventCallback*)

Write the requested number of CAN frames asynchronously to the port This function returns immediately, the specified callback function is called once the buffer has been consumed. The callback function parameter contains the number of frames actually written.

Parameters:

sessionHandle The handle to an existing session

port the port to write the message to

numFrames Number of frames to write

pBuffer Source buffer for the frames

pEventCallback The callback function pointer

Returns:

The status code

3.2.3.747 UeiDaqAPI int UeiDaqWriteCSDBFrame (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numMessages*, tUeiCSDBMessage * *pBuffer*, Int32 * *numMessagesWritten*)

Write the requested number of CSDB message block to the port

Parameters:

sessionHandle The handle to an existing session

port the port to write the message to

numMessages Number of message blocks to write

pBuffer Source buffer for the frames

numMessagesWritten Number of message blocks actually written to the port

Returns:

The status code

3.2.3.748 UeiDaqAPI int UeiDaqWriteDeviceCalibration (DeviceHandle *deviceHandle*, int *channel*, uInt8 *dacMode*, uInt32 *value*, int *saveEEPROM*)

Write/Replace calibration value in the EEPROM area reserved for the specified channel

Parameters:

deviceHandle The handle to a watchdog capable device

channel The channel

dacMode The DAC mode

value The new calibration value

saveEEPROM True to commit all changes to EEPROM

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.749 UeiDaqAPI int UeiDaqWriteDeviceEEPROM (DeviceHandle *deviceHandle*, int *bank*, int *subSystem*, int *size*, uInt8 * *buffer*, tUeiEEPROMArea *area*, int *saveEEPROM*)

Write the device EEPROM content. EEPROM contains informations such as Manufacture date, calibration date and calibration data The EEPROM content is sent as an opaque array of bytes. Each device has a unique way of storing informations in its EEPROM and it is up to the calling application to prepare the buffer to match the EERPROM structure.

Some devices have multiple EEPROM areas. Use the bank parameter to specify which EEPROM you wish to write to

Parameters:

deviceHandle The handle to an existing device

bank The EEPROM bank to write to

subSystem The sub-system to write to

size The number of bytes to write

buffer Buffer containing the new EEPROM content

area The area to access in the EEPROM

saveEEPROM True to commit changes to EEPROM

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.750 UeiDaqAPI int UeiDaqWriteDeviceRAM (DeviceHandle *deviceHandle*, uInt32 *address*, int *size*, uInt8 * *buffer*)

Write to the device RAM. Only for devices that actually have RAM.

Parameters:

deviceHandle The handle to an existing device

address address of the first element to write

size The number of bytes to write

buffer Buffer containing the new value to store in RAM

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.751 UeiDaqAPI int UeiDaqWriteDeviceRegister32 (DeviceHandle *deviceHandle*, uInt32 *offset*, uInt32 *value*)

Write a device register as a 32 bits value.

Parameters:

deviceHandle The handle to an existing device

offset Offset of the register relative to device's base address

value The value to write to the register

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.752 UeiDaqAPI int UeiDaqWriteHDLCFrame (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numBytes*, void * *pBuffer*, Int32 * *numBytesWritten*)

Write the requested number of bytes to the port

Parameters:

sessionHandle The handle to an existing session

port the port to write the message to

numBytes Number of bytes to write

pBuffer Source buffer for the message

numBytesWritten Number of bytes actually written to the port

Returns:

The status code

3.2.3.753 UeiDaqAPI int UeiDaqWriteI2CMasterCommand (SessionHandle *sessionHandle*, Int32 *port*, tUeiI2CMasterCommand * *command*)

Write command for master to transmit.

Parameters:

sessionHandle The handle to an existing session

port The port to write the command to

command The command to write to the master

Returns:

The status code

3.2.3.754 UeiDaqAPI int UeiDaqWriteI2CSlaveData (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numElements*, uInt16 * *pBuffer*, Int32 * *numElementsWritten*)

Writes data to slave FIFO that is transmitted when a master requests data using a read command.

Parameters:

sessionHandle The handle to an existing session

port The port to write the data to

numElements Number of data elements in the buffer

pBuffer Data buffer

numElementsWritten Number of elements actually written to the slave

Returns:

The status code

3.2.3.755 UeiDaqAPI int UeiDaqWriteMessage (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numBytes*, void * *pBuffer*, Int32 * *numBytesWritten*)

Write the requested number of bytes to the port

Parameters:

sessionHandle The handle to an existing session

port the port to write the message to

numBytes Number of bytes to write

pBuffer Source buffer for the message

numBytesWritten Number of bytes actually written to the port

Returns:

The status code

3.2.3.756 UeiDaqAPI int UeiDaqWriteMessageAsync (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numBytes*, void * *pBuffer*, tUeiEventCallback *pEventCallback*)

Write the requested number of bytes asynchronously to the port This function returns immediately, the specified callback function is called once the buffer has been consumed. The callback function parameter contains the number of bytes actually written.

Parameters:

sessionHandle The handle to an existing session

port the port to write the message to

numBytes Number of bytes to write

pBuffer Source buffer for the message

pEventCallback The callback function pointer

Returns:

The status code

3.2.3.757 UeiDaqAPI int UeiDaqWriteMessageInt16 (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numWords*, Int16 * *pBuffer*, Int32 * *numWordsWritten*)

Write the requested number of int16s to the port

Parameters:

sessionHandle The handle to an existing session

port the port to write the message to

numWords Number of words to write

pBuffer Source buffer for the message

numWordsWritten Number of bytes actually written to the port

Returns:

The status code

3.2.3.758 UeiDaqAPI int UeiDaqWriteMIL1553A708ControlFrames (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numFrames*, tUeiMIL1553A708ControlFrame * *pBuffer*, Int32 * *numFramesWritten*)

Write ARINC 708 control frames to the stream. The device associated with this data stream must support ARINC-708

Parameters:

sessionHandle The handle to an existing session

port the port to write to

numFrames Maximum number of frames that can be stored in the buffer

pBuffer Source buffer of the frames

numFramesWritten The number of actual frames written

Returns:

The status code

3.2.3.759 UeiDaqAPI int UeiDaqWriteMIL1553A708DataFrames (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numFrames*, tUeiMIL1553A708DataFrame * *pBuffer*, Int32 * *numFramesWritten*)

Write ARINC 708 data frames to the stream. The device associated with this data stream must support ARINC-708

Parameters:

sessionHandle The handle to an existing session

port the port to write to

numFrames Maximum number of frames that can be stored in the buffer

pBuffer Source buffer of the frames

numFramesWritten The number of actual frames written

Returns:

The status code

3.2.3.760 UeiDaqAPI int UeiDaqWriteMIL1553BCCBDataFrames (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numFrames*, tUeiMIL1553BCCBDataFrame * *pBuffer*, Int32 * *numFramesWritten*)

Write MIL-1553 RT set parameters frames to the stream. The device associated with this data stream must support MIL-1553

Parameters:

sessionHandle The handle to an existing session

port the port to write to

numFrames Maximum number of frames that can be stored in the buffer

pBuffer Source buffer of the frames

numFramesWritten The number of actual frames written

Returns:

The status code

3.2.3.761 UeiDaqAPI int UeiDaqWriteMIL1553BCControlFrames (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numFrames*, tUeiMIL1553BCControlFrame * *pBuffer*, Int32 * *numFramesWritten*)

Write MIL-1553 RT set parameters frames to the stream. The device associated with this data stream must support MIL-1553

Parameters:

sessionHandle The handle to an existing session

port the port to write to

numFrames Maximum number of frames that can be stored in the buffer

pBuffer Source buffer of the frames

numFramesWritten The number of actual frames written

Returns:

The status code

3.2.3.762 UeiDaqAPI int UeiDaqWriteMIL1553BCSchedFrames (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numFrames*, tUeiMIL1553BCSchedFrame * *pBuffer*, Int32 * *numFramesWritten*)

Write MIL-1553 RT set parameters frames to the stream. The device associated with this data stream must support MIL-1553

Parameters:

sessionHandle The handle to an existing session

port the port to write to

numFrames Maximum number of frames that can be stored in the buffer

pBuffer Source buffer of the frames

numFramesWritten The number of actual frames written

Returns:

The status code

3.2.3.763 UeiDaqAPI int UeiDaqWriteMIL1553BusWriterFrames (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numFrames*, tUeiMIL1553BusWriterFrame * *pBuffer*, Int32 * *numFramesWritten*)

Write MIL-1553 bus writer frames to the stream. The device associated with this data stream must support MIL-1553

Parameters:

sessionHandle The handle to an existing session

port the port to write to

numFrames Maximum number of frames that can be stored in the buffer

pBuffer Source buffer of the frames

numFramesWritten The number of actual frames written

Returns:

The status code

3.2.3.764 UeiDaqAPI int UeiDaqWriteMIL1553Frames (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numFrames*, tUeiMIL1553Frame * *pBuffer*, Int32 * *numFramesWritten*)

Write MIL-1553 frames to the stream. The device associated with this data stream must support MIL-1553

Parameters:

sessionHandle The handle to an existing session

port the port to write to

numFrames Maximum number of frames that can be stored in the buffer

pBuffer Source buffer of the frames

numFramesWritten The number of actual frames written

Returns:

The status code

3.2.3.765 UeiDaqAPI int UeiDaqWriteMIL1553FramesAsync (SessionHandle *sessionHandle*, Int32 *port*, Int32 *bufferSize*, tUeiMIL1553Frame * *pBuffer*, tUeiEventCallback *pEventCallback*)

Write MIL-1553 frames to the stream. This function returns immediately, the specified callback function is called once the buffer has been consumed. The callback function parameter contains the number of frames actually written. The device associated with this data stream must support MIL-1553

Parameters:

sessionHandle The handle to an existing session

port the port to write to

bufferSize Maximum number of frames that can be stored in the buffer

pBuffer Source buffer of the frames

pEventCallback The callback function pointer

Returns:

The status code

3.2.3.766 UeiDaqAPI int UeiDaqWriteMIL1553RTControlFrames (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numFrames*, tUeiMIL1553RTControlFrame * *pBuffer*, Int32 * *numFramesWritten*)

Write MIL-1553 RT Control information frames to the stream. The device associated with this data stream must support MIL-1553

Parameters:

sessionHandle The handle to an existing session

port the port to write to

numFrames Maximum number of frames that can be stored in the buffer

pBuffer Source buffer of the frames

numFramesWritten The number of actual frames written

Returns:

The status code

3.2.3.767 UeiDaqAPI int UeiDaqWriteMIL1553RTDataFrames (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numFrames*, tUeiMIL1553RTFrame * *pBuffer*, Int32 * *numFramesWritten*)

Write MIL-1553 RT data area frames to the stream. The device associated with this data stream must support MIL-1553

Parameters:

sessionHandle The handle to an existing session

port the port to write to

numFrames Maximum number of frames that can be stored in the buffer

pBuffer Source buffer of the frames

numFramesWritten The number of actual frames written

Returns:

The status code

3.2.3.768 UeiDaqAPI int UeiDaqWriteMIL1553RTParametersFrames (SessionHandle *sessionHandle*, Int32 *port*, Int32 *numFrames*, tUeiMIL1553RTParametersFrame * *pBuffer*, Int32 * *numFramesWritten*)

Write MIL-1553 RT set parameters frames to the stream. The device associated with this data stream must support MIL-1553

Parameters:

sessionHandle The handle to an existing session

port the port to write to

numFrames Maximum number of frames that can be stored in the buffer

pBuffer Source buffer of the frames

numFramesWritten The number of actual frames written

Returns:

The status code

3.2.3.769 UeiDaqAPI int UeiDaqWriteMux (SessionHandle *sessionHandle*, int *numChannels*, int * *channels*, int * *relayIndices*)

Set mux on a set of channels

Parameters:

sessionHandle The handle to an existing session

numChannels The number of channels to set mux on

channels The list of channels to set mux on

relayIndices The index of the relay to close for each channel in the channel list (0 for relay A, 1 for relay B etc...)

Returns:

The status code

3.2.3.770 UeiDaqAPI int UeiDaqWriteMuxDmm (SessionHandle *sessionHandle*, Int32 *channelNum*, Int32 *relaySelect*, tUeiMuxDmmMode *dmmMode*)

Set mux on a single channel and select output to DMM

Parameters:

sessionHandle The handle to an existing session

channelNum Channel to set

relaySelect Relay to enable. 0 for disable, 1 for relay A, 2 for relay B. Ignored if using UeiMuxDmmMeasRes4 dmmMode

dmmMode Select output to DMM

Returns:

The status code

3.2.3.771 UeiDaqAPI int UeiDaqWriteMuxRaw (SessionHandle *sessionHandle*, int *numValues*, uInt32 * *muxBuffer*)

Set mux on each channel using the same representation than the low-level API Two bits per channels: 00 relays off, 01 relay A ON, 10 relay B on, 11 relay C on [channel 0 mux, channel 1 mux, channel 2 mux, etc...]

Parameters:

sessionHandle The handle to an existing session

numValues Number of values to write

muxBuffer Data buffer to write

Returns:

The status code

3.2.3.772 UeiDaqAPI int UeiDaqWriteMuxRelays (SessionHandle *sessionHandle*, int *numValues*, uInt32 * *relayBuffer*)

Set relays individually (use with care to avoid short circuits and damaging your equipment) One bit per relay [channel 0 relay A, channel 0 relay B, channel 0 relay C, channel 1 relay A, channel 1 relay B, etc...]

Parameters:

sessionHandle The handle to an existing session

numValues Number of values to write

relayBuffer Data buffer to write

Returns:

The status code

3.2.3.773 UeiDaqAPI int UeiDaqWriteRawData16 (SessionHandle *sessionHandle*, Int32 *timeout*, Int32 *numScans*, uInt16 * *pBuffer*)

Write the specified number of 16-bit raw scans to the stream.

Parameters:

sessionHandle The handle to an existing session

timeout The timeout determines the amount of time in ms allowed to read the specified number of scans

numScans Number of scans to write

pBuffer Source buffer for the scans

Returns:

The status code

3.2.3.774 UeiDaqAPI int UeiDaqWriteRawData16Async (SessionHandle *sessionHandle*, Int32 *numScans*, uInt16 * *pBuffer*, tUeiEventCallback *pEventCallback*)

Write 16 bits wide raw scans asynchronously to the stream. This function returns immediately, the specified callback function is called once the buffer has been consumed.

Parameters:

sessionHandle The handle to an existing session

numScans Number of scans to write

pBuffer Source buffer for the scans

pEventCallback The callback function pointer

Returns:

The status code

3.2.3.775 UeiDaqAPI int UeiDaqWriteRawData32 (SessionHandle *sessionHandle*, Int32 *timeout*, Int32 *numScans*, uInt32 * *pBuffer*)

Write the specified number of 32-bit raw scans to the stream.

Parameters:

sessionHandle The handle to an existing session

timeout The timeout determines the amount of time in ms allowed to read the specified number of scans

numScans Number of scans to write

pBuffer Source buffer for the scans

Returns:

The status code

3.2.3.776 UeiDaqAPI int UeiDaqWriteRawData32Async (SessionHandle *sessionHandle*, Int32 *numScans*, uInt32 * *pBuffer*, tUeiEventCallback *pEventCallback*)

Write 32 bits wide raw scans asynchronously to the stream. This function returns immediately, the specified callback function is called once the buffer has been consumed.

Parameters:

sessionHandle The handle to an existing session

numScans Number of scans to write

pBuffer Source buffer for the scans

pEventCallback The callback function pointer

Returns:

The status code

3.2.3.777 UeiDaqAPI int UeiDaqWriteReadDeviceRegisters32 (DeviceHandle *deviceHandle*, uInt32 *writeAddr*, uInt32 *writeSize*, uInt32 * *writeData*, uInt32 *readAddr*, uInt32 *readSize*, uInt32 * *readData*, int *doReadFirst*, int *noIncrement*)

Parameters:

deviceHandle The handle to an existing device

writeAddr Start of address offset that will be written to. Length determined by writeData.Length.

writeSize Number of 32-bit values to write to Device.

writeData Buffer containing data to be written, up to 361 elements.

readAddr Start of address offset that will be read from.

readSize Number of 32-bit values to read from Device.

readData Buffer to store data read from the device, up to 361 elements.

doReadFirst Perform the read before the write.

noIncrement Does not increment to next address (useful for FIFOs).

Returns:

The status code

See also:

UeiDaqGetDeviceHandle (p. 795)

3.2.3.778 UeiDaqAPI int UeiDaqWriteScaledData (SessionHandle *sessionHandle*, Int32 *timeout*, Int32 *numScans*, f64 * *pBuffer*)

Write the specified number of scaled scans to the stream.

Parameters:

sessionHandle The handle to an existing session

timeout The timeout determines the amount of time in ms allowed to read the specified number of scans

numScans Number of scans to write

pBuffer Source buffer for the scans

Returns:

The status code

3.2.3.779 UeiDaqAPI int UeiDaqWriteScaledDataAsync (SessionHandle *sessionHandle*, Int32 *numScans*, f64 * *pBuffer*, tUeiEventCallback *pEventCallback*)

Write scaled scans asynchronously to the stream. This function returns immediately, the specified callback function is called once the buffer has been consumed.

Parameters:

sessionHandle The handle to an existing session

numScans Number of scans to write

pBuffer Source buffer for the scans

pEventCallback The callback function pointer

Returns:

The status code

3.2.3.780 UeiDaqAPI int UeiDaqWriteScheduledARINCRawWords (SessionHandle *sessionHandle*, Int32 *port*, Int32 *firstWord*, Int32 *numWords*, uInt32 * *pBuffer*, Int32 * *numWordsWritten*)

Update scheduler entry with new ARNIC word(s)

Parameters:

sessionHandle The handle to an existing session

port the port to write the message to

firstWord Index of the first word to modify in the scheduler table

numWords Number of words to write in scheduler table

pBuffer Source buffer for the words

numWordsWritten Number of words actually sent to the scheduler

Returns:

The status code

3.2.3.781 UeiDaqAPI int UeiDaqWriteScheduledARINCWords (SessionHandle *sessionHandle*, Int32 *port*, Int32 *firstWord*, Int32 *numWords*, tUeiARINCWord * *pBuffer*, Int32 * *numWordsWritten*)

Update scheduler entry with new raw word(s)

Parameters:

sessionHandle The handle to an existing session

port the port to write the message to

firstWord Index of the first word to modify in the scheduler table

numWords Number of words to write in scheduler table

pBuffer Source buffer for the words

numWordsWritten Number of words actually sent to the scheduler

Returns:

The status code

3.2.3.782 UeiDaqAPI int UeiDaqWriteSSISlaveWords (SessionHandle *sessionHandle*, int *port*, int *grayEncoding*, Int32 *numWords*, uInt32 * *pBuffer*, Int32 * *numWordsWritten*)

Write 32-bit data words to the output stream

Parameters:

sessionHandle The handle to an existing session

port The SSI slave port to write to

grayEncoding true to convert binary values to gray

numWords the number of words to write

pBuffer source buffer

numWordsWritten the actual number of frames sent to the SSI port

Returns:

The status code

3.2.3.783 UeiDaqAPI int UeiDaqWriteSynchroResolverAngles (SessionHandle *sessionHandle*, int *channel*, int *numberOfValues*, uInt32 * *pDelayBuffer*, f64 * *pAngleBuffer*)

Write multiple angles to simulate velocity/acceleration on simulated synchro/resolver Each angle is associated with a delay (in microseconds) The angles are staged to be sent to the device. Actual send happens upon calling UeiDaqSynchroResolverUpdate()

Parameters:

sessionHandle The handle to an existing session

channel the channel to update

numberOfValues numberOfValues in the buffer

pDelayBuffer destination buffer for delays

pAngleBuffer destination buffer for angles

Returns:

The status code

3.3 UeiDaqError.h File Reference

3.3.1 Detailed Description

Error codes reported by UeiDaq Framework when something goes wrong.

Chapter 4

UeiDaq Framework Example Documentation

4.1 AnalogInBuffered.cpp

Buffered Analog Input

This example shows how to acquire a finite set of samples. The acquisition starts immediatly and is timed by a hardware clock.

```
1 #include <stdlib.h>
2 #include <iostream>
3 #include "UeiDaq.h"
4
5 using namespace UeiDaq;
6
7 int main(int argc, char* argv[])
8 {
9    CUeiSession mySs;
10    double* pData = NULL;
11    int count = 0;
12
13    try
14    {
15       // Create 8 analog input channels on a powerdaq board
16       // From now on the session is AI only
17       mySs.CreateAIChannel("simu://Dev0/ai0:7", -10.0, 10.0, UeiAIChannelInputModeDifferential);
18
19       // Configure the session to acquire 1000 scans clocked by internal scan clock
20       mySs.ConfigureTimingForBufferedIO(1000, UeiTimingClockSourceInternal, 10000.0, UeiDigitalEdgeRising, UeiTimingDu
21
22       // The number of frames is automatically calculated
23       // We can override it with the following function calls
24       // mySs.GetDataStream()->SetNumberOfFrames(2);
25
26       // Create a reader object to read data synchronously from the data stream.
27       CUeiAnalogScaledReader reader(mySs.GetDataStream());
28
29       // Allocate buffer to store current frame
30       pData = new double[mySs.GetNumberOfChannels()*mySs.GetDataStream()->GetNumberOfScans()];
31
32       // Start the acquisition, this is optional because the acquisition starts
33       // automatically as soon as the reader starts reading data.
34       mySs.Start();
35
```

```
36       // Acquire 20 frames then stop
37       while(count < 20)
38       {
39          reader.ReadMultipleScans(mySs.GetDataStream()->GetNumberOfScans(), pData);
40
41          for(int i=0; i<mySs.GetNumberOfChannels();i++)
42             std::cout << "ch" << i << " = " << pData[i] << "V, ";
43
44          std::cout << std::endl;
45          count++;
46       }
47
48       mySs.Stop();
49    }
50    catch(CUeiException e)
51    {
52       std::cout << "Error: " << e.GetErrorMessage() << std::endl;
53    }
54
55    if(pData != NULL)
56    {
57       delete[] pData;
58    }
59
60    return 0;
61 }
```

4.2 AnalogInBufferedAsync.cpp

Asynchronous Buffered Analog Input

This example shows how to acquire a finite set of samples. The acquisition starts immediatly and is timed by a hardware clock.

```
1 #include <windows.h>
2 #include <iostream>
3 #include "UeiDaq.h"
4
5 using namespace UeiDaq;
6
7 double* data;
8 int numScans = 50000;
9 double scanRate = 500000;
10 CUeiSession mySs;
11 CUeiAnalogScaledReader* reader;
12
13 class CAnalogInEvent : public IUeiEventListener
14 {
15    void OnEvent(tUeiEvent event, void *param)
16    {
17       if(event == UeiEventFrameDone)
18       {
19          try
20          {
21             // data contains latest samples, do something with it...
22             for(int i=0; i<mySs.GetNumberOfChannels();i++)
23             {
24                std::cout << "ch" << i << " = " << data[i] << "V, ";
25             }
26             std::cout << std::endl;
27
28             // rearm the session to generate the next asynchronous event
29             // once enough new samples are ready
30             reader->ReadMultipleScansAsync(numScans, data);
31          }
32          catch(CUeiException e)
33          {
34             std::cout << "Error: " << e.GetErrorMessage() << std::endl;
35          }
36       }
37       else if(event == UeiEventError)
38       {
39          tUeiError error = (tUeiError)(uintptr_t)param;
40          std::cout << "Error event: " << CUeiException::TranslateError(error) << std::endl;
41       }
42    }
43 };
44
45
46 int main(int argc, char* argv[])
47 {
48    CAnalogInEvent eventListener;
49
50    try
51    {
52       // Create 8 analog input channels on a powerdaq board
53       // From now on the session is AI only
54       mySs.CreateAIChannel("pwrdaq://Dev0/ai0", -10.0, 10.0, UeiAIChannelInputModeDifferential);
55
56       // Configure the session to acquire 1000 scans clocked by internal scan clock
57       mySs.ConfigureTimingForBufferedIO(numScans, UeiTimingClockSourceInternal, scanRate, UeiDigitalEdgeRising, UeiTim
58
59       // Allocate a buffer to hold each acquired frame
```

```
60       data = new double[mySs.GetNumberOfChannels()*mySs.GetDataStream()->GetNumberOfScans()];
61
62       // Create a reader object to read data synchronously.
63       reader = new CUeiAnalogScaledReader(mySs.GetDataStream());
64
65       // Start the acquisition, this is optional because the acquisition starts
66       // automatically as soon as the reader starts reading data.
67       mySs.Start();
68
69       // Configure an asynchronous event handler that will be called
70       // by the reader object each time the required number of scans is available
71       reader->AddEventListener(&eventListener);
72
73       // Arm the reader so that it acquire and store samples in the
74       // data array and call our handler once the array is full
75       reader->ReadMultipleScansAsync(numScans, data);
76
77       // Wait for the user to press the enter key to end the program
78       getchar();
79
80       mySs.Stop();
81    }
82    catch(CUeiException e)
83    {
84       std::cout << "Error: " << e.GetErrorMessage() << std::endl;
85    }
86
87    delete reader;
88    delete[] data;
89
90    return 0;
91 }
```

4.3 AnalogInOneShot_Trig_Ex.cpp

Triggered Buffered Analog Input

This example shows how to acquire a finite set of samples. The acquisition starts when an external trigger occurs and is timed by a hardware clock.

```
1 #include "UeiDaq.h"
2
3 using namespace UeiDaq;
4
5 int main(int argc, char* argv[])
6 {
7    CUeiSession mySs;
8    double data[1000];
9
10    try
11    {
12       // Create 8 analog input channels on a powerdaq board
13       // From now on the session is AI only
14       mySs.CreateAIChannel("pwrdaq://Dev1/ai0:7", -10.0, 10.0, UeiAIChannelInputModeDifferential);
15
16       // Configure the session to acquire 1000 scans clocked by internal scan clock
17       mySs.ConfigureTimingForBufferedIO(1000, UeiTimingClockSourceInternal, 10000.0, UeiDigitalEdgeRising, UeiTimingDu
18
19       // Configure trigger to start the acquisition on an external signal
20       mySs.ConfigureStartTrigger(UeiTriggerSourceExternal, UeiDigitalEdgeRising);
21
22       // The timeout is automatically calculated to give enough time for the required
23       // number of samples per channel to be acquired.
24       // Override it to give more time
25       mySs.GetTiming()->SetTimeout(5000);
26
27       // Create a reader object to read data synchronously.
28       CUeiAnalogScaledReader reader(mySs.GetDataStream());
29
30       // Start the acquisition, this is optional because the acquisition starts
31       // automatically as soon as the reader starts reading data.
32       mySs.Start();
33
34       reader.ReadMultipleScans(1000, data);
35
36       mySs.Stop();
37    }
38    catch(CUeiException e)
39    {
40    }
41
42    return 0;
43 }
```

4.4 AnalogInSingle.cpp

Single Scan Analog Input

This example shows how to do point by point data acquisition.

```
1 #include <iostream>
2 #include "UeiDaq.h"
3
4 using namespace UeiDaq;
5
6 int main(int argc, char* argv[])
7 {
8    CUeiSession mySs;
9    double data[800];
10
11    try
12    {
13       // Create 8 analog input channels on a powerdaq board
14       // From now on the session is AI only
15       mySs.CreateAIChannel("simu://Dev0/ai0:7", -10.0, 10.0, UeiAIChannelInputModeDifferential);
16
17       // By default the timing object is configured for simple I/O so
18       // no need to do anything here.
19       mySs.ConfigureTimingForSimpleIO();
20
21       // Create a reader object to read data synchronously.
22       CUeiAnalogScaledReader reader(mySs.GetDataStream());
23
24       mySs.Start();
25
26       // Read 100 scans
27       for(int i=0; i<100; i++)
28       {
29          reader.ReadSingleScan(&data[i*mySs.GetNumberOfChannels()]);
30          std::cout << "Ch0 = " << data[i*mySs.GetNumberOfChannels()] << std::endl;
31       }
32
33       mySs.Stop();
34    }
35    catch(CUeiException e)
36    {
37       std::cout << "Error: " << e.GetErrorMessage() << std::endl;
38    }
39
40    return 0;
41 }
```

4.5 AnalogOutBuffered.cpp

Buffered Analog Output

This example shows how to generate a finite set of samples. The generation starts immediatly and is timed by a hardware clock.

```
1 #include <iostream>
2 #include "UeiDaq.h"
3
4 using namespace UeiDaq;
5
6
7 int main(int argc, char* argv[])
8 {
9    CUeiSession mySs;
10    double* data;
11
12    try
13    {
14       // Create 2 analog output channels on a powerdaq board
15       // From now on the session is AO only
16       mySs.CreateAOChannel("simu://Dev0/ao0:1", -10.0, 10.0);
17
18       // Configure the session to generate 1000 scans clocked by internal scan clock
19       // The generation will happen only once
20       mySs.ConfigureTimingForBufferedIO(1000, UeiTimingClockSourceInternal, 10000.0, UeiDigitalEdgeRising, UeiTimingDu
21
22       // Allocate a buffer to hold each generated frame
23       data = new double[mySs.GetNumberOfChannels()*mySs.GetDataStream()->GetNumberOfScans()];
24
25       // Create a reader object to read data synchronously.
26       CUeiAnalogScaledWriter writer(mySs.GetDataStream());
27
28       // Fill the buffer before starting generation
29       writer.WriteMultipleScans(1000, data);
30
31       // Start the generation
32       mySs.Start();
33
34       // Wait until all scans are generated
35       while(mySs.IsRunning())
36       {
37       };
38
39       mySs.Stop();
40
41       delete[] data;
42    }
43    catch(CUeiException e)
44    {
45       std::cout << "Error: " << e.GetErrorMessage() << std::endl;
46    }
47
48    return 0;
49 }
```

4.6 AnalogOutBufferedAsync.cpp

Asynchronous Buffered Analog Output

This example shows how to asynchronously generate a finite set of samples. The generation starts immediatly and is timed by a hardware clock.

```
1 #include <windows.h>
2 #include <iostream>
3 #include <math.h>
4 #include "UeiDaq.h"
5
6 using namespace UeiDaq;
7
8 CUeiSession mySs;
9 double* data;
10 CUeiAnalogScaledWriter* writer;
11
12 void GenerateSinWave(double* pBuffer, int nbChannels, int nbSamplePerChannel, int iteration)
13 {
14    int amplitude = (iteration % 10 + 1);
15
16    for(int i=0; i<nbSamplePerChannel; i++)
17    {
18       for(int j=0; j<nbChannels; j++)
19       {
20          pBuffer[i*nbChannels+j] = amplitude * sin(2*3.1415*(j+1)*i/nbSamplePerChannel);
21       }
22    }
23 }
24
25 class CAnalogOutEvent : public IUeiEventListener
26 {
27    void OnEvent(tUeiEvent event, void *param)
28    {
29       static int count = 0;
30       if(event == UeiEventFrameDone)
31       {
32          try
33          {
34             std::cout << "event #" << count++ << " received" << std::endl;
35
36             // If regeneration is turned off we can refresh the buffer to generate
37             // new data continuously, this call blocks until there is enough
38             // room in the buffer to write the required number of samples
39             writer->WriteMultipleScansAsync(1000, data);
40
41             // Create a new set of data to be generated next time we get an event
42             GenerateSinWave(data, mySs.GetNumberOfChannels(), 1000, 0);
43          }
44          catch(CUeiException e)
45          {
46             std::cout << "Error: " << e.GetErrorMessage() << std::endl;
47          }
48       }
49       else if(event == UeiEventError)
50       {
51          tUeiError error = (tUeiError)(uintptr_t)param;
52          std::cout << "Error: " << CUeiException::TranslateError(error) << std::endl;
53       }
54    }
55 };
56
57 int main(int argc, char* argv[])
58 {
59    try
```

```
60    {
61       // Create 2 analog output channels on a powerdaq board
62       // From now on the session is AO only
63       mySs.CreateAOChannel("pwrdaq://Dev0/ao0:7", -10.0, 10.0);
64
65       // Configure the session to generate 1000 scans clocked by internal scan clock
66       mySs.ConfigureTimingForBufferedIO(1000, UeiTimingClockSourceInternal, 10000.0, UeiDigitalEdgeRising, UeiTimingDu
67
68       // Allocate a buffer to hold each generated frame
69       data = new double[mySs.GetNumberOfChannels()*mySs.GetDataStream()->GetNumberOfScans()];
70
71       // Create a reader object to read data synchronously.
72       writer = new CUeiAnalogScaledWriter(mySs.GetDataStream());
73
74       // Configure an asynchronous event handler that will be called
75       // by the writer object each time the required number of scans has been generated
76       CAnalogOutEvent eventListener;
77       writer->AddEventListener(&eventListener);
78
79       // Start the generation
80       mySs.Start();
81
82       // Fill the buffer
83       GenerateSinWave(data, mySs.GetNumberOfChannels(), 1000, 0);
84       writer->WriteMultipleScansAsync(1000, data);
85
86       // Data transfer is done asynchronously
87       // Press the Enter key to end the generation
88       std::cout << "Press 'Enter' to stop the generation." << std::endl;
89       getchar();
90
91       mySs.Stop();
92
93       delete writer;
94       delete[] data;
95    }
96    catch(CUeiException e)
97    {
98       std::cout << "Error: " << e.GetErrorMessage() << std::endl;
99    }
100
101    return 0;
102 }
```

4.7 AnalogOutSingle.cpp

Point by point Analog Output

This example shows how to do point by point data generation.

```
1 #include <iostream>
2 #include <cmath>
3 #include "UeiDaq.h"
4
5 using namespace UeiDaq;
6
7 void GenerateSinWave(double* pBuffer, int nbChannels, int nbSamplePerChannel, int iteration)
8 {
9    int amplitude = (iteration % 10 + 1);
10
11    for(int i=0; i<nbSamplePerChannel; i++)
12    {
13       for(int j=0; j<nbChannels; j++)
14       {
15          pBuffer[i*nbChannels+j] = amplitude * sin(2*3.1415*(j+1)*i/nbSamplePerChannel);
16       }
17    }
18 }
19
20 int main(int argc, char* argv[])
21 {
22    CUeiSession mySs;
23    double* data;
24
25    try
26    {
27       // Create 2 analog output channels on a powerdaq board
28       // From now on the session is AO only
29       mySs.CreateAOChannel("pwrdaq://Dev0/ao0:1", -10.0, 10.0);
30
31       mySs.ConfigureTimingForSimpleIO();
32
33       // Create a reader object to read data synchronously.
34       CUeiAnalogScaledWriter writer(mySs.GetDataStream());
35
36       // Allocate a buffer to hold data to generate
37       data = new double[mySs.GetNumberOfChannels()*100];
38
39       GenerateSinWave(data, mySs.GetNumberOfChannels(), 100, 0);
40
41       // Generates all points in the buffer
42       for(int i=0; i<100; i++)
43       {
44          std::cout << "Generating scan " << i << std::endl;
45          writer.WriteSingleScan(&data[i*2]);
46       }
47
48       delete[] data;
49    }
50    catch(CUeiException e)
51    {
52       std::cout << "Error: " << e.GetErrorMessage() << std::endl;
53    }
54
55    return 0;
56 }
```

4.8 BufferedEventCounting.cpp

Buffered Event counting

This example shows how to count digital edges using a counter-timer The event counting is timed by a hardware clock

```
1 #include <iostream>
2 #include "UeiDaq.h"
3
4 using namespace UeiDaq;
5
6 int main(int argc, char* argv[])
7 {
8    CUeiSession mySs;
9    unsigned short countedEvents[1000];
10
11    try
12    {
13       // Create one counter input channel on a powerdaq board to count digital
14       // events applied on the input of the counter.
15       // From now on the session is CI only
16       mySs.CreateCIChannel("pwrdaq://Dev0/ci0",
17                            UeiCounterSourceInput,
18                            UeiCounterModeCountEvents,
19                            UeiCounterGateInternal,
20                            1,
21                            false);
22
23       // Configure the session to acquire 1000 counts clocked by internal clock
24       mySs.ConfigureTimingForBufferedIO(1000, UeiTimingClockSourceInternal, 10000.0, UeiDigitalEdgeRising, UeiTimingDu
25
26       // Create a reader object to read data synchronously.
27       CUeiCounterReader reader(mySs.GetDataStream());
28
29       // Start counting
30       mySs.Start();
31
32       // Read 100 samples
33       reader.ReadMultipleScans(100, countedEvents);
34
35       mySs.Stop();
36    }
37    catch(CUeiException e)
38    {
39       std::cout << "Error: " << e.GetErrorMessage() << std::endl;
40    }
41
42    return 0;
43 }
```

4.9 DigitalInEvent.cpp

Event counting

This example shows how to configure a board to send an event when the digital state of an input line changes

```
1 #include <iostream>
2 #include "UeiDaq.h"
3
4 using namespace UeiDaq;
5
6 int main(int argc, char* argv[])
7 {
8    CUeiSession mySs;
9
10
11    try
12    {
13       // Create 1 digital input channel on a powerdaq board
14       // From now on the session is DI only
15       CUeiDIChannel* pChan = mySs.CreateDIChannel("simu://Dev0/di0");
16
17       // Set the mask on the channel to respond to events on every digital lines
18       pChan->SetEdgeMask(0xFFFF);
19
20       // Configure the session to detect signal change on line 0 of the first port
21       // in the port list, the session will detect edges continuously.
22       // The data stream will store 1000 readings at a time
23       mySs.ConfigureTimingForEdgeDetection(UeiDigitalEdgeRising);
24
25       mySs.GetTiming()->SetTimeout(5000);
26
27       // Create a reader and a listener object to receive events asynchronously.
28       CUeiDigitalReader reader(mySs.GetDataStream());
29
30       // Start the session
31       mySs.Start();
32
33       // Read 100 events
34       for(int i=0; i<100; i++)
35       {
36          unsigned short data[1];
37          reader.ReadSingleScan(data);
38
39          std::cout << "Received Digital event #" << i << ": " << std::hex << data[0] << std::endl;
40       }
41
42       mySs.Stop();
43    }
44    catch(CUeiException e)
45    {
46       std::cout << "Error: " << e.GetErrorMessage() << std::endl;
47    }
48
49    return 0;
50 }
51
```

4.10 EventCount.cpp

Event counting

This example shows how to count digital edges using a counter-timer

```
1 #include <iostream>
2 #include "UeiDaq.h"
3
4 using namespace UeiDaq;
5
6 int main(int argc, char* argv[])
7 {
8    CUeiSession mySs;
9    unsigned short countedEvents;
10
11    try
12    {
13       // Create one counter input channel on a powerdaq board to count digital
14       // events applied on the input of the counter.
15       // From now on the session is CI only
16       mySs.CreateCIChannel("pwrdaq://Dev0/ci0",
17                            UeiCounterSourceInput,
18                            UeiCounterModeCountEvents,
19                            UeiCounterGateInternal,
20                            1,
21                            false);
22
23       mySs.ConfigureTimingForSimpleIO();
24
25       // Create a reader object to read data synchronously.
26       CUeiCounterReader reader(mySs.GetDataStream());
27
28       // Start counting
29       mySs.Start();
30
31       // Read 100 values and return
32       for(int i=0; i<100; i++)
33       {
34          reader.ReadSingleScan(&countedEvents);
35          std::cout << "Read " << countedEvents << " events" << std::endl;
36       }
37
38       mySs.Stop();
39    }
40    catch(CUeiException e)
41    {
42       std::cout << "Error: " << e.GetErrorMessage() << std::endl;
43    }
44
45    return 0;
46 }
```

4.11 GeneratePulseTrain.cpp

Pulse Train Generation

This example shows how to generate a pulse train using a counter-timer

```
1 #include <Windows.h>
2 #include <iostream>
3 #include "UeiDaq.h"
4
5 using namespace UeiDaq;
6
7 int main(int argc, char* argv[])
8 {
9    CUeiSession mySs;
10
11    try
12    {
13       // Create one counter output channel on a powerdaq board to generate a pulse train
14       // using the internal clock as a reference. Each pulse will be low for 50000 ticks and
15       // high for another 50000 ticks
16       // From now on the session is CI only
17       mySs.CreateCOChannel("pdna://192.168.100.3/Dev1/co0", UeiCounterSourceClock, UeiCounterModeGeneratePulseTrain,
18                            UeiCounterGateInternal, 20000, 50000, 1, false);
19
20       mySs.ConfigureTimingForSimpleIO();
21
22       CUeiCounterWriter writer(mySs.GetDataStream());
23
24       // Start generating the pulse train for one second
25       mySs.Start();
26
27       for (int i = 0; i < 100; i++)
28       {
29          Sleep(500);
30          uInt32 pwm[2];
31
32          // Ticks are specified in 66Mhz period intervals
33
34          // low time for 10us * i
35          pwm[0] = 10 * i * 66;
36          // high time for 20us * i
37          pwm[1] = 5 * i * 66;
38
39          writer.WriteSingleScan(pwm);
40       }
41
42       mySs.Stop();
43    }
44    catch(CUeiException e)
45    {
46       std::cout << "Error: " << e.GetErrorMessage() << std::endl;
47    }
48
49    return 0;
50 }
```

4.12 MeasureFrequency.cpp

Event counting

This example shows how to measure a digital signal frequency using a counter-timer

```
1 #include <iostream>
2 #include "UeiDaq.h"
3
4 using namespace UeiDaq;
5
6 int main(int argc, char* argv[])
7 {
8    CUeiSession ciSs;
9    uInt16* datas = NULL;   // for devices with 16 bit or less counter resolution
10    uInt32* datal = NULL;   // for devices with counter resolution greater than 16 bit
11
12
13    try
14    {
15       // Create one counter input channel on a powerdaq board to measure the grequency
16       // of the signal applied to the input of the selected counter.
17       // From now on the session is CI only
18       ciSs.CreateCIChannel("pdna://192.168.15.200/Dev5/ci0",
19                            UeiCounterSourceInput,
20                            UeiCounterModeMeasurePeriod,
21                            UeiCounterGateInternal,
22                            1,
23                            false);
24
25       ciSs.ConfigureTimingForSimpleIO();
26
27       // Configure debounce filter for each channel.
28       // This step is optional and is only needed if your signal
29       // is noisy and you want to filter out unwanted spikes
30       for(int ch=0; ch<ciSs.GetNumberOfChannels(); ch++)
31       {
32          CUeiCIChannel* pChannel = dynamic_cast<CUeiCIChannel*>(ciSs.GetChannel(ch));
33
34          // filter out pulses shorter than 1ms
35          pChannel->SetMinimumSourcePulseWidth(1.0);
36       }
37
38       // Measurement data will be stored in a 16 bits or 32 bits integer buffer
39       // depending on the counter resolution
40       // let's allocate a buffer big enough to hold one value for each configured port
41       if(ciSs.GetDevice()->GetCIResolution() <= 16)
42       {
43          datas = new uInt16[ciSs.GetNumberOfChannels()];
44       }
45       else
46       {
47          datal = new uInt32[ciSs.GetNumberOfChannels()];
48       }
49
50       // Create a reader object to read data synchronously.
51       CUeiCounterReader reader(ciSs.GetDataStream());
52
53       // Start period measurement
54       ciSs.Start();
55
56       // Read period 100 times
57       for(int i=0; i<100; i++)
58       {
59          double period;
60
```

```
61          if(datas != NULL)
62          {
63             reader.ReadSingleScan(datas);
64             std::cout << "period width = " << datas[0] << " ticks" << std::endl;
65          }
66          else
67          {
68             reader.ReadSingleScan(datal);
69
70             // Use counter base frequency to convert reading to secs.
71             // CT-601 base frequency is 66MHz
72             period = datal[0]/66000000.0;
73             std::cout << "period width = " << datal[0] << " ticks, "
74                       << period << " secs, " << 1.0/period << "Hz" << std::endl;
75          }
76       }
77
78       ciSs.Stop();
79    }
80    catch(CUeiException e)
81    {
82       std::cout << "Error: " << e.GetErrorMessage() << std::endl;
83    }
84
85    if(datas != NULL)
86    {
87       delete[] datas;
88    }
89    if(datal != NULL)
90    {
91       delete[] datal;
92    }
93
94    return 0;
95 }
```

Index