2015 Test beam Run Control

Generated by Doxygen 1.8.9.1

Thu Apr 23 2015 15:25:38

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

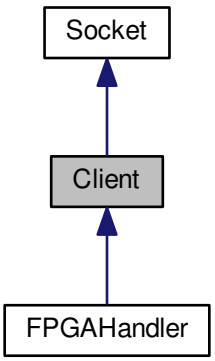
Data Structure Documentation

3.1 Client Class Reference

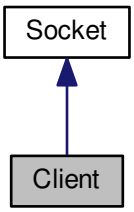
Base client object for the socket.

`#include <Client.h>`

Inheritance diagram for Client:



Collaboration diagram for Client:



Public Member Functions

- Client ()

  General void client constructor.
- Client (int port)

  Bind a socket client to a given port.
- virtual ~Client ()
- bool Connect ()

  Bind this client to the socket.
- void Disconnect ()

  Unbind this client from the socket.
- void Send (const Message &m) const

  Send a message to the master through the socket.
- void Receive ()

  Receive a socket message from the master.
- virtual void ParseMessage (const SocketMessage &m)

  Parse a SocketMessage received from the master.
- virtual SocketType GetType () const

  Socket actor type retrieval method.

Additional Inherited Members

3.1.1 Detailed Description

Base client object for the socket.

Client object used by the server to send/receive commands from the messenger/broadcaster.

Author

Laurent Forthomme laurent.forthomme@cern.ch

Date

24 Mar 2015

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Client::Client ( ) [inline]

General void client constructor.

3.1.2.2 Client::Client (int *port*)

Bind a socket client to a given port.

3.1.2.3 virtual Client::~Client ( ) [virtual]

3.1.3 Member Function Documentation

3.1.3.1 bool Client::Connect ( )

Bind this client to the socket.

3.1.3.2 void Client::Disconnect ( )

Unbind this client from the socket.

3.1.3.3 virtual SocketType Client::GetType ( ) const [inline], [virtual]

Socket actor type retrieval method.

Reimplemented in FPGAHandler.

3.1.3.4 virtual void Client::ParseMessage (const SocketMessage & *m*) [inline], [virtual]

Parse a SocketMessage received from the master.

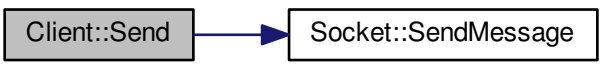
3.1.3.5 void Client::Receive ( )

Receive a socket message from the master.

3.1.3.6 void Client::Send (const Message & *m*) const [inline]

Send a message to the master through the socket.

Here is the call graph for this function:



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

- include/Client.h

3.2 Exception Class Reference

A simple exception handler.

`#include <Exception.h>`

Public Member Functions

- Exception (const char *from, std::string desc, ExceptionType type=Undefined, const int id=0)
- Exception (const char *from, const char *desc, ExceptionType type=Undefined, const int id=0)
- ~Exception ()
- std::string From () const
- int ErrorNumber () const
- std::string Description () const
- ExceptionType Type () const
- std::string TypeString () const
- void Dump (std::ostream &os=std::cerr) const

3.2.1 Detailed Description

A simple exception handler.

Author

Laurent Forthomme `laurent.forthomme@cern.ch`

Date

24 Mar 2015

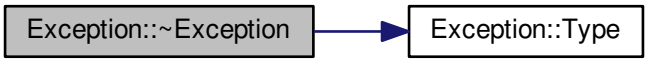
3.2.2 Constructor & Destructor Documentation

3.2.2.1 Exception::Exception (const char * *from*, std::string *desc*, ExceptionType *type* = `Undefined`, const int *id* = `0`) `[inline]`

3.2.2.2 Exception::Exception (const char * *from*, const char * *desc*, ExceptionType *type* = `Undefined`, const int *id* = `0`) `[inline]`

3.2.2.3 Exception::~Exception ( ) `[inline]`

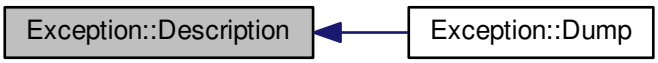
Here is the call graph for this function:



3.2.3 Member Function Documentation

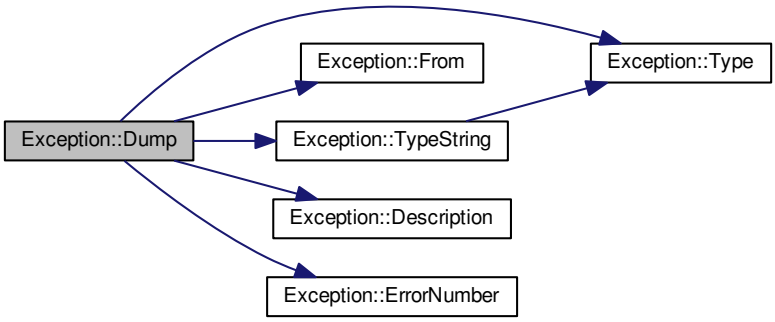
3.2.3.1 std::string Exception::Description ( ) const [inline]

Here is the caller graph for this function:



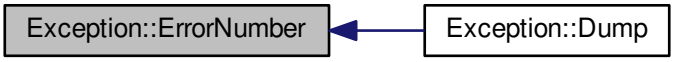
3.2.3.2 void Exception::Dump (std::ostream & *os* = std::cerr) const [inline]

Here is the call graph for this function:



3.2.3.3 int Exception::ErrorNumber ( ) const [inline]

Here is the caller graph for this function:

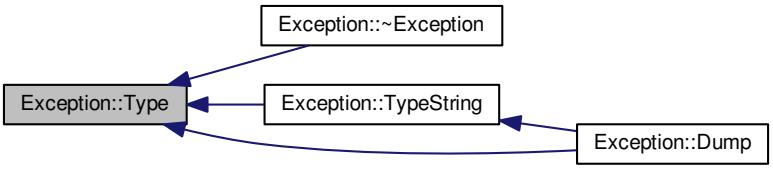


3.2.3.4 std::string Exception::From ( ) const [inline]

Here is the caller graph for this function:

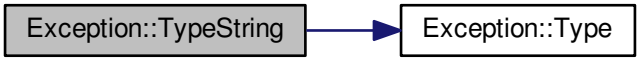
3.2.3.5 ExceptionType Exception::Type ( ) const [inline]

Here is the caller graph for this function:

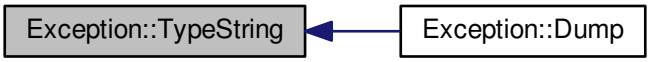


3.2.3.6 std::string Exception::TypeString ( ) const [inline]

Here is the call graph for this function:



Here is the caller graph for this function:



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

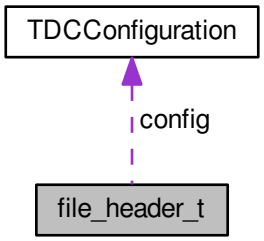
- include/Exception.h

3.3 file_header_t Struct Reference

Header to the output files.

`#include <FPGAHandler.h>`

Collaboration diagram for file_header_t:



Data Fields

- uint32_t magic
- uint32_t run_id
- uint32_t spill_id
- TDCConfiguration config

3.3.1 Detailed Description

Header to the output files.

General header to store in each collected data file for offline readout. It enable any reader to retrieve the run/spill number, as well as the HPTDC configuration during data collection.

Author

Laurent Forthomme laurent.forthomme@cern.ch

Date

14 Apr 2015

3.3.2 Field Documentation

3.3.2.1 TDCConfiguration file_header_t::config

3.3.2.2 uint32_t file_header_t::magic

3.3.2.3 uint32_t file_header_t::run_id

3.3.2.4 uint32_t file_header_t::spill_id

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

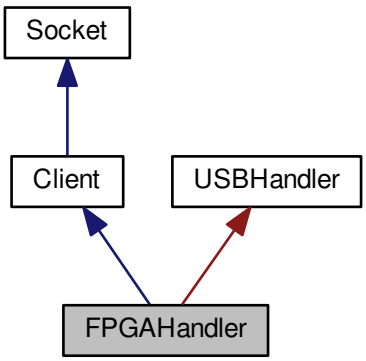
- include/FPGAHandler.h

3.4 FPGAHandler Class Reference

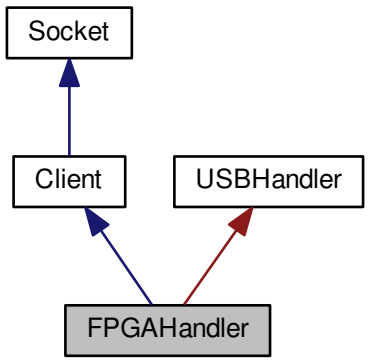
Driver for timing detectors' FPGA readout.

`#include <FPGAHandler.h>`

Inheritance diagram for FPGAHandler:



Collaboration diagram for FPGAHandler:



Public Member Functions

- FPGAHandler (int port, const char *dev)

  Bind to a FPGA through the USB protocol, and to the socket.
- virtual ~FPGAHandler ()
- void OpenFile ()

  Open an output file to store header/HPTDC events.
- void CloseFile ()

  Close a previously opened output file used to store header/HPTDC events.
- std::string GetFilename () const

  Retrieve the file name used to store data collected from the FPGA.
- void SetConfiguration (const TDCConfiguration &c)

  Submit the HPTDC setup word as a TDCConfiguration object.
- TDCConfiguration GetConfiguration ()

  Retrieve the HPTDC setup word as a TDCConfiguration object.
- void ReadBuffer ()
- SocketType GetType () const

  Socket actor type retrieval method.

Additional Inherited Members

3.4.1 Detailed Description

Driver for timing detectors' FPGA readout.

Main driver for a homebrew FPGA designed for the timing detectors' HPTDC chip readout.

Author

Laurent Forthomme laurent.forthomme@cern.ch

Date

14 Apr 2015

3.4.2 Constructor & Destructor Documentation

3.4.2.1 FPGAHandler::FPGAHandler (int *port,* const char * *dev*)

Bind to a FPGA through the USB protocol, and to the socket.

3.4.2.2 virtual FPGAHandler::~FPGAHandler ( ) [virtual]

3.4.3 Member Function Documentation

3.4.3.1 void FPGAHandler::CloseFile ( )

Close a previously opened output file used to store header/HPTDC events.

3.4.3.2 TDCConfiguration FPGAHandler::GetConfiguration ( ) [inline]

Retrieve the HPTDC setup word as a TDCConfiguration object.

3.4.3.3 std::string FPGAHandler::GetFilename ( ) const [inline]

Retrieve the file name used to store data collected from the FPGA.

3.4.3.4 SocketType FPGAHandler::GetType ( ) const [inline], [virtual]

Socket actor type retrieval method.

Reimplemented from Client.

3.4.3.5 void FPGAHandler::OpenFile ( )

Open an output file to store header/HPTDC events.

3.4.3.6 void FPGAHandler::ReadBuffer ( )

3.4.3.7 void FPGAHandler::SetConfiguration (const TDCConfiguration & *c*) [inline]

Submit the HPTDC setup word as a TDCConfiguration object.

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

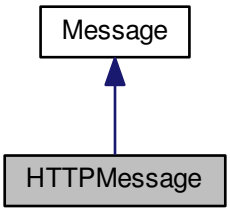
- include/FPGAHandler.h

3.5 HTTPMessage Class Reference

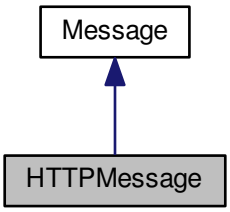
Message to be transmitted through a WebSocket protocol.

`#include <HTTPMessage.h>`

Inheritance diagram for HTTPMessage:



Collaboration diagram for HTTPMessage:



Public Member Functions

- HTTPMessage (WebSocket ∗ws, Message m, MessageAction a)
- HTTPMessage (WebSocket ∗ws, const char ∗msg, MessageAction a)
- void Decode ()
- void Encode ()
- MessageKey GetKey () const
- void Dump (std::ostream &os=std::cout) const

Additional Inherited Members

3.5.1 Detailed Description

Message to be transmitted through a WebSocket protocol.

Type of message compatible to the transmission through a WebSocket protocol. It enables a direct conversion of standards from any socket message format used elsewhere in this code using the *MessageAction* statement.

Author

Laurent Forthomme laurent.forthomme@cern.ch

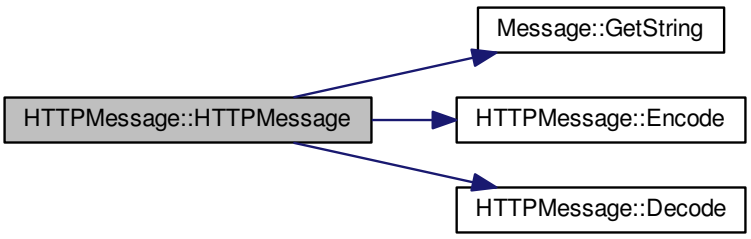
Date

1 Apr 2015

3.5.2 Constructor & Destructor Documentation

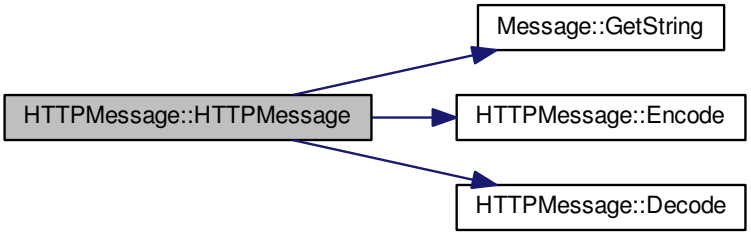
3.5.2.1 HTTPMessage::HTTPMessage (WebSocket * *ws*, Message *m*, MessageAction *a*) [inline]

Here is the call graph for this function:



3.5.2.2 HTTPMessage::HTTPMessage (WebSocket * *ws*, const char * *msg*, MessageAction *a*) [inline]

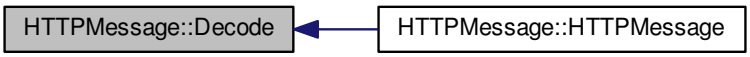
Here is the call graph for this function:



3.5.3 Member Function Documentation

3.5.3.1 void HTTPMessage::Decode ( ) [inline]

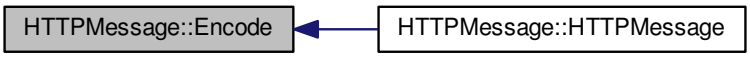
Here is the caller graph for this function:



3.5.3.2 void HTTPMessage::Dump (std::ostream & *os* = std::cout) const [inline]

3.5.3.3 void HTTPMessage::Encode ( ) [inline]

Here is the caller graph for this function:



3.5.3.4 MessageKey HTTPMessage::GetKey ( ) const [inline]

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

- include/HTTPMessage.h

3.6 ListenerInfo Struct Reference

Information on a socket's listener.

`#include <Messenger.h>`

Data Fields

- std::string name
- SocketType type

3.6.1 Detailed Description

Information on a socket's listener.

Structure handling its name and type for any listener/client to be used in the socket management parts of this code.

3.6.2 Field Documentation

3.6.2.1 std::string ListenerInfo::name

3.6.2.2 SocketType ListenerInfo::type

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

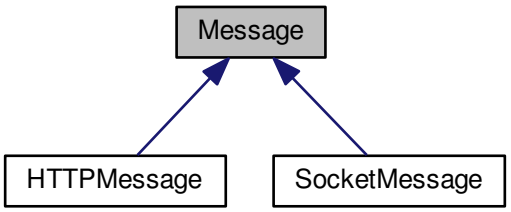
- include/Messenger.h

3.7 Message Class Reference

Base socket message type.

`#include <Message.h>`

Inheritance diagram for Message:



Public Member Functions

- Message ()

  Void message constructor.
- Message (const char *msg)

  Construct a message from a string.
- Message (std::string msg)

  Construct a message from a string.
- virtual ~Message ()
- MessageKey GetKey () const

  Placeholder for the MessageKey retrieval method.
- std::string GetString () const

  Retrieve the string carried by this message as a whole.
- bool IsFromWeb () const

  Extract from any message its potential arrival from a WebSocket protocol.
- void Dump (std::ostream &os=std::cout) const

Protected Attributes

- std::string fString

3.7.1 Detailed Description

Base socket message type.

Base handler for messages to be transmitted through the socket

Author

Laurent Forthomme laurent.forthomme@cern.ch

Date

6 Apr 2015

3.7.2 Constructor & Destructor Documentation

3.7.2.1 Message::Message ( ) [inline]

Void message constructor.

3.7.2.2 Message::Message (const char * *msg*) [inline]

Construct a message from a string.

3.7.2.3 Message::Message (std::string *msg*) [inline]

Construct a message from a string.

3.7.2.4 virtual Message::~Message ( ) [inline], [virtual]

3.7.3 Member Function Documentation

3.7.3.1 void Message::Dump (std::ostream & *os* = std::cout) const [inline]

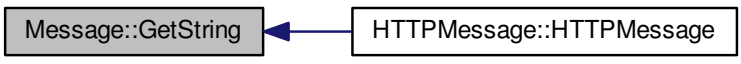
3.7.3.2 MessageKey Message::GetKey ( ) const [inline]

Placeholder for the MessageKey retrieval method.

3.7.3.3 std::string Message::GetString ( ) const [inline]

Retrieve the string carried by this message as a whole.

Here is the caller graph for this function:



3.7.3.4 bool Message::IsFromWeb ( ) const [inline]

Extract from any message its potential arrival from a WebSocket protocol.

3.7.4 Field Documentation

3.7.4.1 std::string Message::fString [protected]

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

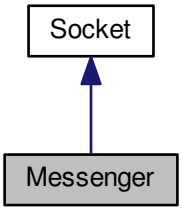
- include/Message.h

3.8 Messenger Class Reference

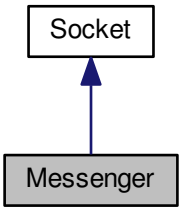
Base master object for the socket.

`#include <Messenger.h>`

Inheritance diagram for Messenger:



Collaboration diagram for Messenger:



Public Member Functions

- Messenger ()

Build a void master object or socket actor.

- Messenger (int port)

  Build a master object to control the socket.
- ~Messenger ()
- bool Connect ()

  Connect the master to the socket.
- void Disconnect ()

  Remove the master and destroy the socket.
- void Send (const Message &m, int sid) const

  Send any type of message to any client.
- void Receive ()

  Handle a message reception from a client.
- void Broadcast (const Message &m) const

  Emit a message to all clients connected through the socket.
- SocketType GetType () const

  Socket actor type retrieval method.

Additional Inherited Members

3.8.1 Detailed Description

Base master object for the socket.

Messenger/broadcaster object used by the server to send/receive commands from the clients/listeners.

Author

Laurent Forthomme laurent.forthomme@cern.ch

Date

23 Mar 2015

3.8.2 Constructor & Destructor Documentation

3.8.2.1 Messenger::Messenger ( )

Build a void master object or socket actor.

3.8.2.2 Messenger::Messenger (int *port*)

Build a master object to control the socket.

3.8.2.3 Messenger::~Messenger ( )

3.8.3 Member Function Documentation

3.8.3.1 void Messenger::Broadcast (const Message & *m*) const

Emit a message to all clients connected through the socket.

Parameters

in	*m*	Message to transmit

3.8.3.2 bool Messenger::Connect ( )

Connect the master to the socket.

Connect this master to the socket for clients to be able to bind.

3.8.3.3 void Messenger::Disconnect ( )

Remove the master and destroy the socket.

Remove this master from the socket, thus disconnecting automatically the clients connected.

3.8.3.4 SocketType Messenger::GetType ( ) const [inline]

Socket actor type retrieval method.

3.8.3.5 void Messenger::Receive ( )

Handle a message reception from a client.

3.8.3.6 void Messenger::Send (const Message & *m*, int *sid*) const [inline]

Send any type of message to any client.

Parameters

in	*m*	Message to transmit
in	*sid*	Unique identifier of the client on this socket

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

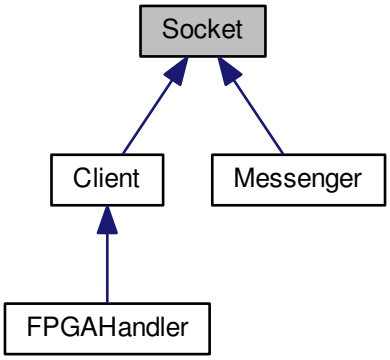
- include/Messenger.h

3.9 Socket Class Reference

Base socket object from which clients/master from a socket inherit.

`#include <Socket.h>`

Inheritance diagram for Socket:



Public Member Functions

- Socket ()
- Socket (int port)
- virtual ~Socket ()
- void Stop ()

  Terminates the socket and all attached communications.
- void SetPort (int port)
- int GetPort () const

  Retrieve the port used for this socket.
- void AcceptConnections (Socket &socket)

  Accept connection from a client.
- void SelectConnections ()
- void SetSocketId (int sid)
- int GetSocketId () const
- SocketType GetSocketType (int sid) const
- bool IsWebSocket (int sid) const
- void DumpConnected () const

Protected Member Functions

- bool Start ()

  Start the socket.
- void Bind ()

  Bind a name to a socket.
- void PrepareConnection ()
- void Listen (int maxconn)

  Listen to incoming messages.
- void SendMessage (Message message, int id=-1) const

  Send a message on a socket.
- Message FetchMessage (int id=-1) const

  Receive a message from a socket.

Protected Attributes

- int fPort
- char fBuffer [MAX_WORD_LENGTH]
- SocketCollection fSocketsConnected
- fd_set fMaster

  Master file descriptor list.
- fd_set fReadFds

  Temp file descriptor list for select()

3.9.1 Detailed Description

Base socket object from which clients/master from a socket inherit.

General object providing all useful method to connect/bind/send/receive information through system sockets.

Author

Laurent Forthomme laurent.forthomme@cern.ch

Date

23 Mar 2015

3.9.2 Constructor & Destructor Documentation

3.9.2.1 Socket::Socket ( ) [inline]

3.9.2.2 Socket::Socket (int *port*)

3.9.2.3 virtual Socket::~Socket ( ) [virtual]

3.9.3 Member Function Documentation

3.9.3.1 void Socket::AcceptConnections (Socket & *socket*)

Accept connection from a client.

Set the socket to accept connections any client transmitting through the socket

Parameters

in,out	*socket*	Master/client object to enable on the socket

3.9.3.2 void Socket::Bind ( ) [protected]

Bind a name to a socket.

Returns

Success of the operation

3.9.3.3 void Socket::DumpConnected ( ) const

3.9.3.4 Message Socket::FetchMessage (int *id* = -1) const [protected]

Receive a message from a socket.

Returns

Received message as a std::string

3.9.3.5 int Socket::GetPort ( ) const [inline]

Retrieve the port used for this socket.

3.9.3.6 int Socket::GetSocketId ( ) const [inline]

3.9.3.7 SocketType Socket::GetSocketType (int *sid*) const [inline]

Here is the caller graph for this function:

3.9.3.8 bool Socket::IsWebSocket (int *sid*) const [inline]

Here is the call graph for this function:

3.9.3.9 void Socket::Listen (int *maxconn*) [protected]

Listen to incoming messages.

Set the socket to listen to any message coming from outside

3.9.3.10 void Socket::PrepareConnection ( ) [protected]

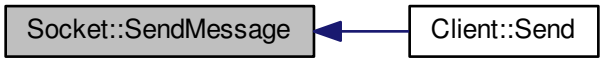
3.9.3.11 void Socket::SelectConnections ( )

Register all open file descriptors to read their communication through the socket

3.9.3.12 void Socket::SendMessage (Message *message*, int *id* = -1) const [protected]

Send a message on a socket.

Here is the caller graph for this function:



3.9.3.13 void Socket::SetPort (int *port*) [inline]

3.9.3.14 void Socket::SetSocketId (int *sid*) [inline]

3.9.3.15 bool Socket::Start ( ) [protected]

Start the socket.

Launch all mandatory operations to set the socket to be used

Returns

Success of the operation

3.9.3.16 void Socket::Stop ( )

Terminates the socket and all attached communications.

3.9.4 Field Documentation

3.9.4.1 char Socket::fBuffer[MAX_WORD_LENGTH] [protected]

3.9.4.2 fd_set Socket::fMaster [protected]

Master file descriptor list.

3.9.4.3 int Socket::fPort [protected]

3.9.4.4 fd_set Socket::fReadFds [protected]

Temp file descriptor list for select()

3.9.4.5 SocketCollection Socket::fSocketsConnected [protected]

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

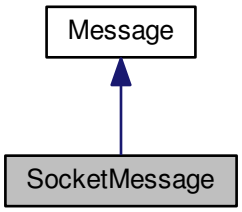
- include/Socket.h

3.10 SocketMessage Class Reference

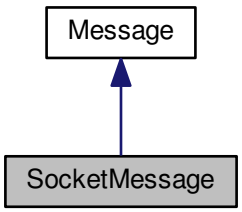
Socket-passed message type.

`#include <SocketMessage.h>`

Inheritance diagram for SocketMessage:



Collaboration diagram for SocketMessage:



Public Member Functions

- SocketMessage ()
- SocketMessage (const Message &msg)
- SocketMessage (const char *msg_s)
- SocketMessage (std::string msg_s)
- SocketMessage (const MessageKey &key)

  Construct a socket message out of a key.
- SocketMessage (const MessageKey &key, const char *value)

Construct a socket message out of a key and a string-type value.

- SocketMessage (const MessageKey &key, std::string value)

  Construct a socket message out of a key and a string-type value.

- SocketMessage (const MessageKey &key, const int value)

  Construct a socket message out of a key and an integer-type value.

- SocketMessage (const MessageKey &key, const float value)

  Construct a socket message out of a key and a float-type value.

- SocketMessage (const MessageKey &key, const double value)

  Construct a socket message out of a key and a double precision-type value.

- SocketMessage (MessageMap msg_m)

  Construct a socket message out of a map of key/string-type value.

- ~SocketMessage ()
- void SetKeyValue (const MessageKey &key, const char *value)

  String-valued message.

- void SetKeyValue (const MessageKey &key, int int_value)

  Send an integer-valued message.

- void SetKeyValue (const MessageKey &key, float float_value)

  Float-valued message.

- void SetKeyValue (const MessageKey &key, double double_value)

  Double-valued message.

- std::string GetString () const

  Extract the whole key:value message.

- MessageKey GetKey () const

  Extract the message's key.

- std::string GetValue () const

  Extract the message's string value.

- int GetIntValue () const

  Extract the message's integer value.

- VectorValue GetVectorValue () const

  Extract the message's vector of string value.

- void Dump (std::ostream &os=std::cout) const

Additional Inherited Members

3.10.1 Detailed Description

Socket-passed message type.

Author

Laurent Forthomme laurent.forthomme@cern.ch

Date

26 Mar 2015

3.10.2 Constructor & Destructor Documentation

3.10.2.1 SocketMessage::SocketMessage ( ) [inline]

3.10.2.2 SocketMessage::SocketMessage (const Message & *msg*) [inline]

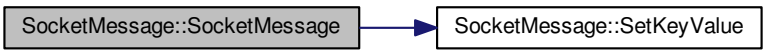
3.10.2.3 SocketMessage::SocketMessage (const char * *msg_s*) [inline]

3.10.2.4 SocketMessage::SocketMessage (std::string *msg_s*) [inline]

3.10.2.5 SocketMessage::SocketMessage (const MessageKey & *key*) [inline]

Construct a socket message out of a key.

Here is the call graph for this function:



3.10.2.6 SocketMessage::SocketMessage (const MessageKey & *key*, const char * *value*) [inline]

Construct a socket message out of a key and a string-type value.

Here is the call graph for this function:

3.10.2.7 SocketMessage::SocketMessage (const MessageKey & *key*, std::string *value*) [inline]

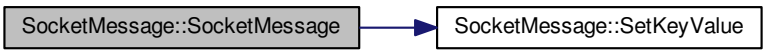
Construct a socket message out of a key and a string-type value.

Here is the call graph for this function:

3.10.2.8 SocketMessage::SocketMessage (const MessageKey & *key*, const int *value*) [inline]

Construct a socket message out of a key and an integer-type value.

Here is the call graph for this function:



3.10.2.9 SocketMessage::SocketMessage (const MessageKey & *key*, const float *value*) [inline]

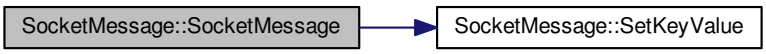
Construct a socket message out of a key and a float-type value.

Here is the call graph for this function:

3.10.2.10 SocketMessage::SocketMessage (const MessageKey & *key*, const double *value*) [inline]

Construct a socket message out of a key and a double precision-type value.

Here is the call graph for this function:



3.10.2.11 SocketMessage::SocketMessage (MessageMap *msg_m*) [inline]

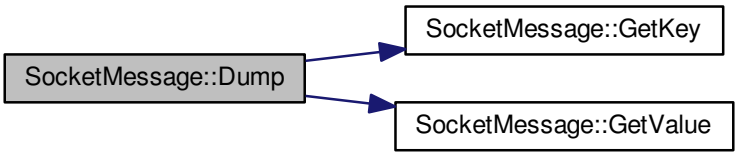
Construct a socket message out of a map of key/string-type value.

3.10.2.12 SocketMessage::~SocketMessage ( ) [inline]

3.10.3 Member Function Documentation

3.10.3.1 void SocketMessage::Dump (std::ostream & *os* = `std::cout`) const [inline]

Here is the call graph for this function:



3.10.3.2 int SocketMessage::GetIntValue ( ) const [inline]

Extract the message's integer value.

3.10.3.3 MessageKey SocketMessage::GetKey ( ) const [inline]

Extract the message's key.

Here is the caller graph for this function:

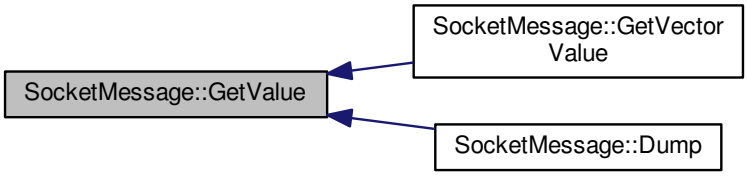
3.10.3.4 std::string SocketMessage::GetString ( ) const [inline]

Extract the whole key:value message.

3.10.3.5 std::string SocketMessage::GetValue ( ) const [inline]

Extract the message's string value.

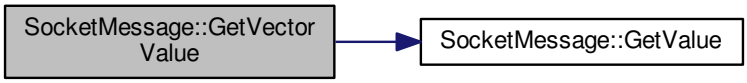
Here is the caller graph for this function:



3.10.3.6 VectorValue SocketMessage::GetVectorValue ( ) const [inline]

Extract the message's vector of string value.

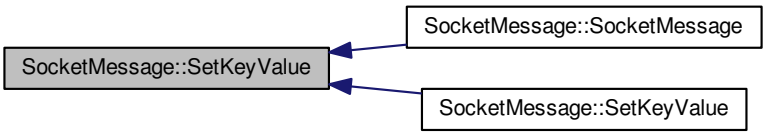
Here is the call graph for this function:



3.10.3.7 void SocketMessage::SetKeyValue (const MessageKey & *key*, const char * *value*) [inline]

String-valued message.

Here is the caller graph for this function:



3.10.3.8 void SocketMessage::SetKeyValue (const MessageKey & *key*, int *int_value*) [inline]

Send an integer-valued message.

Here is the call graph for this function:

3.10.3.9 void SocketMessage::SetKeyValue (const MessageKey & *key,* float *float_value*) [inline]

Float-valued message.

Here is the call graph for this function:

3.10.3.10 void SocketMessage::SetKeyValue (const MessageKey & *key,* double *double_value*) [inline]

Double-valued message.

Here is the call graph for this function:

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

- include/SocketMessage.h

3.11 TDCConfiguration Class Reference

Setup word to be sent to the HPTDC chip.

`#include <TDCConfiguration.h>`

Public Types

- enum EdgeResolution {
  E_100ps =0, E_200ps, E_400ps, E_800ps,
  E_1p6ns, E_3p12ns, E_6p25ns, E_12p5ns }
- enum DeadTime { DT_5ns =0, DT_10ns, DT_30ns, DT_100ns }
- enum WidthResolution {
  W_100ps =0, W_200ps, W_400ps, W_800ps,
  W_1p6ns, W_3p2ns, W_6p25ns, W_12p5ns,
  W_25ns, W_50ns, W_100ns, W_200ns,
  W_400ns, W_800ns }
- enum EnabledError {
  VernierError =0x1, CoarseError =0x2, ChannelSelectError =0x4, L1BufferParityError =0x8,
  TriggerFIFOParityError =0x10, TriggerMatchingError =0x20, ReadoutFIFOParityError =0x40, ReadoutState↩
  Error =0x80,
  SetupParityError =0x100, ControlParityError =0x200, JTAGInstructionParityError =0x400 }

Public Member Functions

- TDCConfiguration ()
- virtual ~TDCConfiguration ()
- void SetWord (const unsigned int i, const word_t word)

  Set one bit(s) subset in the setup word.
- word_t GetWord (const unsigned int i) const

  Retrieve one subset from the setup word.
- uint8_t GetNumWords () const

  Number of words in the configuration.
- void SetEnableErrorMark (bool em)

  Mark events with error if global error signal is set.
- bool GetEnableErrorMark () const
- void SetEnableErrorBypass (bool eb)

  Bypass TDC chip if global error signal is set.
- bool GetEnableErrorBypass () const
- void SetEnableError (const uint16_t &err)

  Enable internal error types for generation of global error signals.
- uint16_t GetEnableError () const
- void SetEnableSerial (bool es)

  Enable of serial read-out (otherwise parallel read-out)
- bool GetEnableSerial () const
- void SetEnableJTAGReadout (bool jr)

  Enable of read-out via JTAG.
- bool GetEnableJTAGReadout () const
- void SetEdgeResolution (const EdgeResolution r)
- EdgeResolution GetEdgeResolution () const
- void SetMaxEventSize (int sz)

  Set the maximum number of hits per event.
- uint8_t GetMaxEventSize () const

  Extract the maximum number of hits per event.
- void SetRejectFIFOFull (bool rej=true)

  Reject hits when readout FIFO full.
- bool GetRejectFIFOFull () const

  Are hits rejected when readout FIFO is full?
- void SetEnableReadoutOccupancy (const bool ro=true)

Enable the readout of buffer occupancies for each event (for debugging purposes)

- bool GetEnableReadoutOccupancy () const
- void SetEnableReadoutSeparator (const bool ro=true)

  Enable the readout of separators for each event (for debugging purposes, valid if readout of occupancies is enabled)
- bool GetEnableReadoutSeparator () const
- void SetTriggerCountOffset (uint16_t tco)

  Set offset for the trigger time tag counter.
- uint16_t GetTriggerCountOffset () const

  Extract trigger time tag count offset.
- void SetChannelOffset (int channel, uint16_t offset)
- uint16_t GetChannelOffset (int channel) const
- void SetAllChannelsOffset (uint16_t offset)
- void SetCoarseCountOffset (uint16_t cco)

  Set offset for the coarse time counter.
- uint16_t GetCoarseCountOffset () const

  Extract offset for the coarse time counter.
- void SetDLLAdjustment (int tap, uint8_t adj)

  Set the DLL taps adjustments with a resolution of ∼10 ps.
- uint8_t GetDLLAdjustment (int tap) const
- void SetAllTapsDLLAdjustment (uint8_t adj)
- void SetRCAdjustment (int tap, uint8_t adj)
- uint8_t GetRCAdjustment (int tap)
- void SetWidthResolution (const WidthResolution r)
- WidthResolution GetWidthResolution () const
- void SetVernierOffset (const uint8_t vo)

  Set the offset in vernier decoding.
- uint8_t GetVernierOffset () const

  Extract the offset in vernier decoding.
- void SetDeadTime (const DeadTime dt)
- DeadTime GetDeadTime () const
- void SetLeadingMode (const bool lead=true)

  Enable the detection of leading edges.
- bool GetLeadingMode () const

  Extract the status for the detection of leading edges.
- void SetTrailingMode (const bool trail=true)

  Enable/disable the detection of trailing edges.
- bool GetTrailingMode () const

  Extract the status for the detection of trailing edges.
- void SetTriggerMatchingMode (const bool trig=true)
- bool GetTriggerMatchingMode () const
- void SetEdgesPairing (const bool pair=true)
- bool GetEdgesPairing () const
- void SetConstantValues ()

  Ensure that the critical constant values are properly set in the setup word.
- uint16_t GetTriggerLatency () const

  Effective trigger latency in number of clock cycles (when no counter roll-over is used)
- void Dump (int verb=1, std::ostream &os=std::cout) const

3.11.1 Detailed Description

Setup word to be sent to the HPTDC chip.

Object handling the configuration word provided by/to the HPTDC chip

Author

Laurent Forthomme laurent.forthomme@cern.ch

Date

16 Apr 2015

3.11.2 Member Enumeration Documentation

3.11.2.1 enum TDCConfiguration::DeadTime

Enumerator

DT_5ns
DT_10ns
DT_30ns
DT_100ns

3.11.2.2 enum TDCConfiguration::EdgeResolution

Enumerator

E_100ps
E_200ps
E_400ps
E_800ps
E_1p6ns
E_3p12ns
E_6p25ns
E_12p5ns

3.11.2.3 enum TDCConfiguration::EnabledError

Enumerator

VernierError
CoarseError
ChannelSelectError
L1BufferParityError
TriggerFIFOParityError
TriggerMatchingError
ReadoutFIFOParityError
ReadoutStateError
SetupParityError
ControlParityError
JTAGInstructionParityError

3.11.2.4 enum TDCConfiguration::WidthResolution

Enumerator

W_100ps
W_200ps
W_400ps
W_800ps
W_1p6ns
W_3p2ns
W_6p25ns
W_12p5ns
W_25ns
W_50ns
W_100ns
W_200ns
W_400ns
W_800ns

3.11.3 Constructor & Destructor Documentation

3.11.3.1 TDCConfiguration::TDCConfiguration ( )

3.11.3.2 virtual TDCConfiguration::~TDCConfiguration ( ) [inline], [virtual]

3.11.4 Member Function Documentation

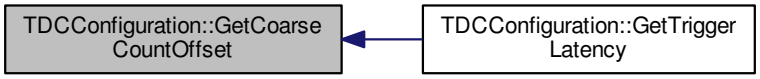
3.11.4.1 void TDCConfiguration::Dump (int *verb* = `1`, std::ostream & *os* = `std::cout`) const

3.11.4.2 uint16_t TDCConfiguration::GetChannelOffset (int *channel*) const [inline]

3.11.4.3 uint16_t TDCConfiguration::GetCoarseCountOffset ( ) const [inline]

Extract offset for the coarse time counter.

Here is the caller graph for this function:



3.11.4.4 DeadTime TDCConfiguration::GetDeadTime ( ) const [inline]

3.11.4.5 uint8_t TDCConfiguration::GetDLLAdjustment (int *tap*) const [inline]

3.11.4.6 EdgeResolution TDCConfiguration::GetEdgeResolution ( ) const [inline]

3.11.4.7 bool TDCConfiguration::GetEdgesPairing ( ) const [inline]

3.11.4.8 uint16_t TDCConfiguration::GetEnableError ( ) const [inline]

3.11.4.9 bool TDCConfiguration::GetEnableErrorBypass ( ) const [inline]

3.11.4.10 bool TDCConfiguration::GetEnableErrorMark ( ) const [inline]

3.11.4.11 bool TDCConfiguration::GetEnableJTAGReadout ( ) const [inline]

3.11.4.12 bool TDCConfiguration::GetEnableReadoutOccupancy ( ) const [inline]

Here is the caller graph for this function:

3.11.4.13 bool TDCConfiguration::GetEnableReadoutSeparator ( ) const [inline]

3.11.4.14 bool TDCConfiguration::GetEnableSerial ( ) const [inline]

3.11.4.15 bool TDCConfiguration::GetLeadingMode ( ) const [inline]

Extract the status for the detection of leading edges.

3.11.4.16 uint8_t TDCConfiguration::GetMaxEventSize ( ) const [inline]

Extract the maximum number of hits per event.

3.11.4.17 uint8_t TDCConfiguration::GetNumWords ( ) const [inline]

Number of words in the configuration.

Return the number of words making up the full configuration word.

3.11.4.18 uint8_t TDCConfiguration::GetRCAdjustment (int *tap*) [inline]

3.11.4.19 bool TDCConfiguration::GetRejectFIFOFull ( ) const [inline]

Are hits rejected when readout FIFO is full?

Extract whether or not hits are rejected once FIFO is full.

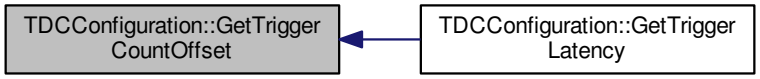
3.11.4.20 bool TDCConfiguration::GetTrailingMode ( ) const [inline]

Extract the status for the detection of trailing edges.

3.11.4.21 uint16_t TDCConfiguration::GetTriggerCountOffset ( ) const [inline]

Extract trigger time tag count offset.

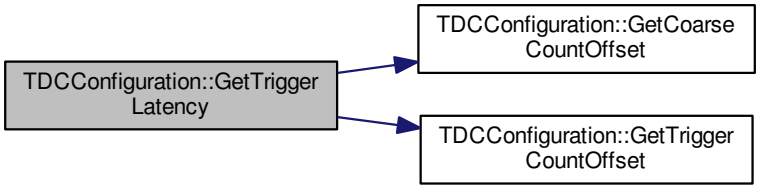
Here is the caller graph for this function:



3.11.4.22 uint16_t TDCConfiguration::GetTriggerLatency ( ) const [inline]

Effective trigger latency in number of clock cycles (when no counter roll-over is used)

Here is the call graph for this function:



3.11.4.23 bool TDCConfiguration::GetTriggerMatchingMode ( ) const [inline]

3.11.4.24 uint8_t TDCConfiguration::GetVernierOffset ( ) const [inline]

Extract the offset in vernier decoding.

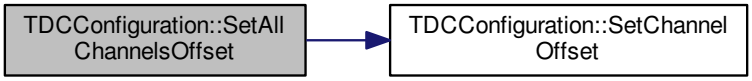
3.11.4.25 WidthResolution TDCConfiguration::GetWidthResolution ( ) const [inline]

3.11.4.26 word_t TDCConfiguration::GetWord (const unsigned int *i*) const [inline]

Retrieve one subset from the setup word.

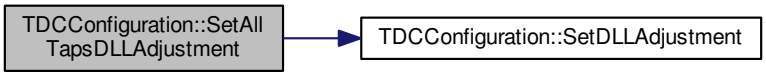
3.11.4.27 void TDCConfiguration::SetAllChannelsOffset (uint16_t *offset*) [inline]

Here is the call graph for this function:



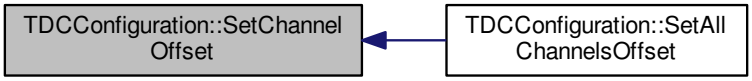
3.11.4.28 void TDCConfiguration::SetAllTapsDLLAdjustment (uint8_t *adj*) [inline]

Here is the call graph for this function:



3.11.4.29 void TDCConfiguration::SetChannelOffset (int *channel,* uint16_t *offset*) [inline]

Here is the caller graph for this function:



3.11.4.30 void TDCConfiguration::SetCoarseCountOffset (uint16_t *cco*) [inline]

Set offset for the coarse time counter.

3.11.4.31 void TDCConfiguration::SetConstantValues ( ) [inline]

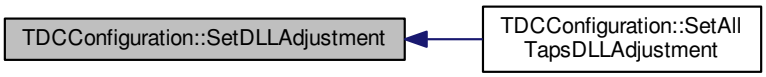
Ensure that the critical constant values are properly set in the setup word.

3.11.4.32 void TDCConfiguration::SetDeadTime (const DeadTime *dt*) [inline]

3.11.4.33 void TDCConfiguration::SetDLLAdjustment (int *tap*, uint8_t *adj*) [inline]

Set the DLL taps adjustments with a resolution of ∼10 ps.

Here is the caller graph for this function:



3.11.4.34 void TDCConfiguration::SetEdgeResolution (const EdgeResolution *r*) [inline]

3.11.4.35 void TDCConfiguration::SetEdgesPairing (const bool *pair* = true) [inline]

3.11.4.36 void TDCConfiguration::SetEnableError (const uint16_t & *err*) [inline]

Enable internal error types for generation of global error signals.

3.11.4.37 void TDCConfiguration::SetEnableErrorBypass (bool *eb*) [inline]

Bypass TDC chip if global error signal is set.

3.11.4.38 void TDCConfiguration::SetEnableErrorMark (bool *em*) [inline]

Mark events with error if global error signal is set.

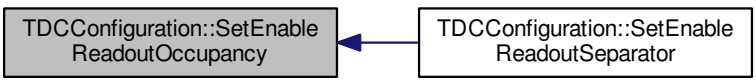
3.11.4.39 void TDCConfiguration::SetEnableJTAGReadout (bool *jr*) [inline]

Enable of read-out via JTAG.

3.11.4.40 void TDCConfiguration::SetEnableReadoutOccupancy (const bool *ro* = true) [inline]

Enable the readout of buffer occupancies for each event (for debugging purposes)

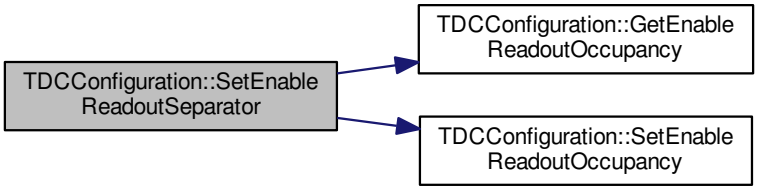
Here is the caller graph for this function:



3.11.4.41 void TDCConfiguration::SetEnableReadoutSeparator (const bool *ro* = `true`) [inline]

Enable the readout of separators for each event (for debugging purposes, valid if readout of occupancies is enabled)

Here is the call graph for this function:



3.11.4.42 void TDCConfiguration::SetEnableSerial (bool *es*) [inline]

Enable of serial read-out (otherwise parallel read-out)

3.11.4.43 void TDCConfiguration::SetLeadingMode (const bool *lead* = `true`) [inline]

Enable the detection of leading edges.

3.11.4.44 void TDCConfiguration::SetMaxEventSize (int *sz*) [inline]

Set the maximum number of hits per event.

Set the maximum number of hits that can be recorded for each event. It is always rounded to the next power of 2 (in the range 0-128), and if lower than 0 or bigger than 128 then set to unimited.

3.11.4.45 void TDCConfiguration::SetRCAdjustment (int *tap*, uint8_t *adj*) [inline]

3.11.4.46 void TDCConfiguration::SetRejectFIFOFull (bool *rej* = `true`) [inline]

Reject hits when readout FIFO full.

Set whether or not hits are rejected once FIFO is full.

3.11.4.47 void TDCConfiguration::SetTrailingMode (const bool *trail* = `true`) [inline]

Enable/disable the detection of trailing edges.

3.11.4.48 void TDCConfiguration::SetTriggerCountOffset (uint16_t *tco*) [inline]

Set offset for the trigger time tag counter.

3.11.4.49 void TDCConfiguration::SetTriggerMatchingMode (const bool *trig* = `true`) [inline]

3.11.4.50 void TDCConfiguration::SetVernierOffset (const uint8_t *vo*) [inline]

Set the offset in vernier decoding.

3.11.4.51 void TDCConfiguration::SetWidthResolution (const WidthResolution *r*) [inline]

3.11.4.52 void TDCConfiguration::SetWord (const unsigned int *i*, const word_t *word*) [inline]

Set one bit(s) subset in the setup word.

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

- include/TDCConfiguration.h

3.12 TDCEvent Class Reference

HPTDC event parser.

`#include <TDCEvent.h>`

Public Types

- enum EventType {
  Invalid =-1, GroupHeader =0, GroupTrailer, TDCHeader,
  TDCTrailer, LeadingEdge, TrailingEdge, Error,
  Debug }

Public Member Functions

- TDCEvent (const uint32_t &word)
- virtual ~TDCEvent ()
- EventType GetType () const

  Type of packet read out from the TDC.
- unsigned int GetTDCId () const

  Programmed identifier of master TDC.
- uint16_t GetEventId () const

  Event identifier from event counter.
- uint16_t GetWordCount () const

  Total number of words in event (including headers and trailers)
- uint16_t GetBunchId () const

  Bunch identifier of trigger (or trigger time tag)
- uint32_t GetLeadingTime (bool pair=false) const

  Leading edge measurement in programmed time resolution.
- uint8_t GetWidth () const

  Width of pulse in programmed time resolution.
- uint32_t GetTrailingTime () const

  Trailing edge measurement in programmed time resolution.
- uint16_t GetErrorFlags () const

  Return error flags if an error condition has been detected.

3.12.1 Detailed Description

HPTDC event parser.

Object enabling to decipher any measurement/error/debug event returned by the HPTDC chip

Author

Laurent Forthomme laurent.forthomme@cern.ch

Date

20 Apr 2015

3.12.2 Member Enumeration Documentation

3.12.2.1 enum TDCEvent::EventType

Enumerator

Invalid

GroupHeader

GroupTrailer

TDCHeader

TDCTrailer

LeadingEdge

TrailingEdge

Error

Debug

3.12.3 Constructor & Destructor Documentation

3.12.3.1 TDCEvent::TDCEvent (const uint32_t & *word*) [inline]

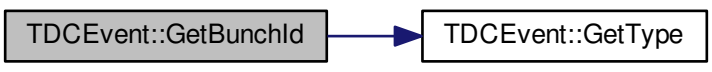
3.12.3.2 virtual TDCEvent::~TDCEvent ( ) [inline], [virtual]

3.12.4 Member Function Documentation

3.12.4.1 uint16_t TDCEvent::GetBunchId ( ) const [inline]

Bunch identifier of trigger (or trigger time tag)

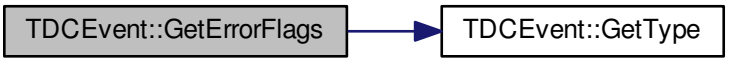
Here is the call graph for this function:



3.12.4.2 uint16_t TDCEvent::GetErrorFlags ( ) const [inline]

Return error flags if an error condition has been detected.

Here is the call graph for this function:



3.12.4.3 uint16_t TDCEvent::GetEventId ( ) const [inline]

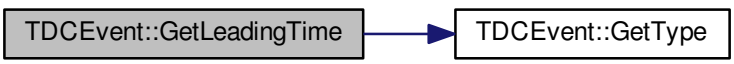
Event identifier from event counter.

Here is the call graph for this function:

3.12.4.4 uint32_t TDCEvent::GetLeadingTime (bool *pair* = false) const [inline]

Leading edge measurement in programmed time resolution.

Here is the call graph for this function:



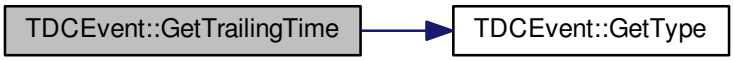
3.12.4.5 unsigned int TDCEvent::GetTDCId ( ) const [inline]

Programmed identifier of master TDC.

3.12.4.6 uint32_t TDCEvent::GetTrailingTime ( ) const [inline]

Trailing edge measurement in programmed time resolution.

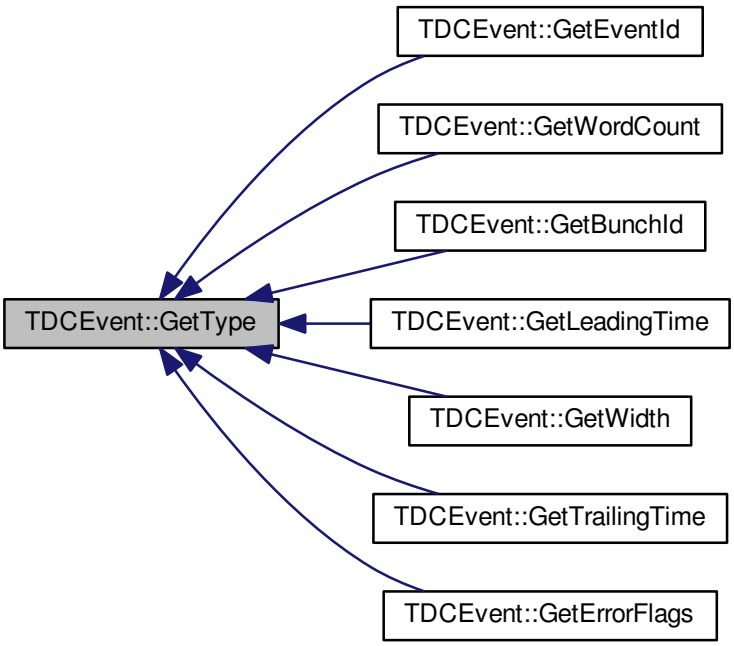
Here is the call graph for this function:



3.12.4.7 EventType TDCEvent::GetType ( ) const [inline]

Type of packet read out from the TDC.

Here is the caller graph for this function:



3.12.4.8 uint8_t TDCEvent::GetWidth ( ) const [inline]

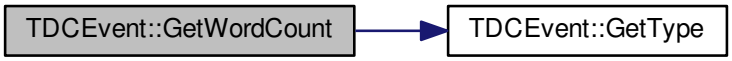
Width of pulse in programmed time resolution.

Here is the call graph for this function:

3.12.4.9 uint16_t TDCEvent::GetWordCount ( ) const [inline]

Total number of words in event (including headers and trailers)

Here is the call graph for this function:



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

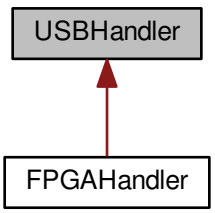
- include/TDCEvent.h

3.13 USBHandler Class Reference

Generic USB communication handler.

`#include <USBHandler.h>`

Inheritance diagram for USBHandler:



Public Member Functions

- USBHandler (const char ∗dev)
- virtual ∼USBHandler ()
- void Init ()
- void DumpDevice (libusb_device ∗dev, int verb=1, std::ostream &out=std::cout)

Protected Member Functions

- void Write (uint32_t word, uint8_t size) const

  Write a word to the USB device.
- uint32_t Fetch (uint8_t size) const

  Receive a word from the USB device.

3.13.1 Detailed Description

Generic USB communication handler.

Date

21 Apr 2015

Author

Laurent Forthomme laurent.forthomme@cern.ch

3.13.2 Constructor & Destructor Documentation

3.13.2.1 USBHandler::USBHandler (const char ∗ *dev*)

3.13.2.2 virtual USBHandler::∼USBHandler ( ) [inline], [virtual]

3.13.3 Member Function Documentation

3.13.3.1 void USBHandler::DumpDevice (libusb_device ∗ *dev*, int *verb* = 1, std::ostream & *out* = std::cout)

3.13.3.2 uint32_t USBHandler::Fetch (uint8_t *size*) const [inline], [protected]

Receive a word from the USB device.

3.13.3.3 void USBHandler::Init ( )

3.13.3.4 void USBHandler::Write (uint32_t *word*, uint8_t *size*) const [inline], [protected]

Write a word to the USB device.

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

- include/USBHandler.h

Index