DAQ: a ROOT based Data AcQuisition platform for debugging embedded devices

0.1

Generated by Doxygen 1.5.6

Tue Sep 1 19:01:08 2009

Contents

Chapter 1

DAQ: a ROOT based Data AcQuisition platform for debugging embedded devices

1.1 Motivation

In Embedded Systems the validation of software solutions is quite difficult with respect to high-end devices for the lack of ordinary debugging tools like display, file system, etc. The communication towards a PC permits to debug the software code making use of tools available in high-end devices only (awk, grep, find, etc.) Moreover it is quite beneficial to implement data serialization in order to assess the performances of the embedded system firmware (implementing some kind of logic) with respect to full-fledged algorithms available in high end devices only. We propose this framework including serialization and post-processing virtues coming from the ROOT framework developed at CERN with standardization and extendibility feautures coming from the xml language. We propose an example to test the performances of a full-customed firmware for onboard image processing.

1.2 Software architecture

The architecture of DAQ is based essentially on 4 blocks:

- **xml parser** to interpret the communication specs (physical specs, handshake feature, data pattern...);
- **data reader** to read raw data directly from the API functions;
- **data formatter** to format data following the given data pattern;
- **ROOT serializer** to store data into a repository (ROOT file) on the filesystem.

The working principle is based on the feature extraction from xml file and the creation of 2 data structures: one to define the reader tasks and one to describe the packet content in terms of fields (vector of metadata). For such puroposes the structs **SDef** (p. 116) and **SMData** (p. 121) are defined. So the logic follows these steps:

- start of the communication (initialization and handshake)

- syncronization with the stream (by lenght or by the start and end symbols)

- raw data reading

- raw data processing (decoding and fragmentation in messages prior to interpretation)

- data formatting following the metadata definition

The output of this system is a vector of vector of **SData (p. 114)**, a structure that contains data formatted plus the information needed for the case of casting and processing.

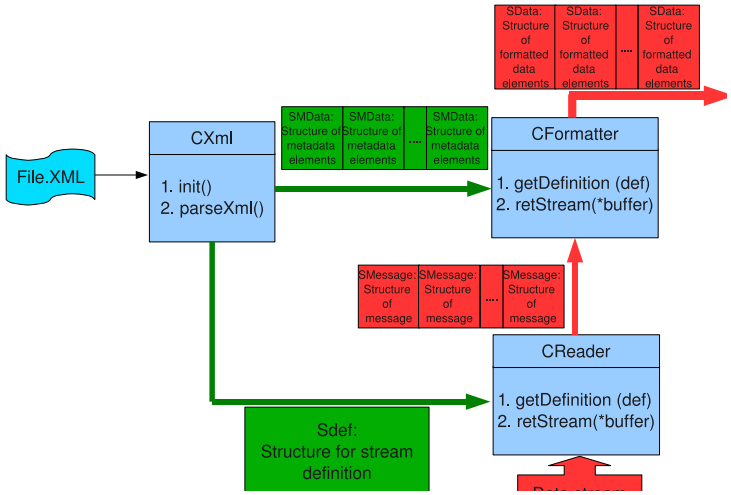


Figure 1.1: DAQ architecture

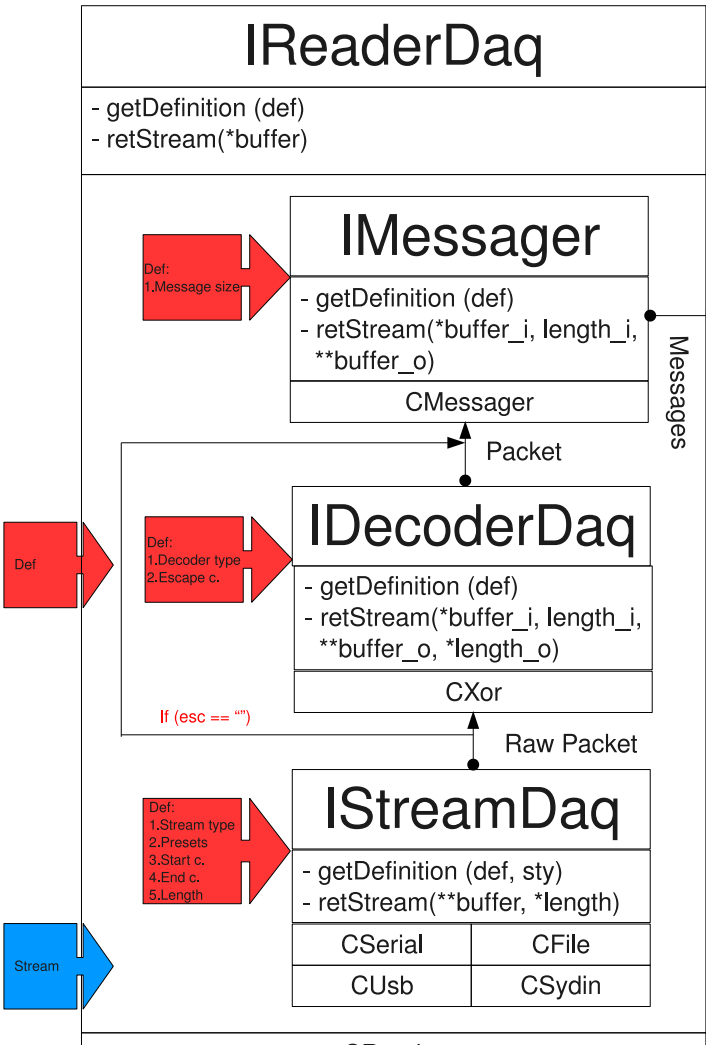


Figure 1.2: Reader structure

The formatted data can be stored in a structured repository called ROOT designed by CERN to store data from experiments in High Energy Physics. ROOT Framework implements procedures for data debugging permitting post-processing (off-line) of "structured" data: this capability is very valuable whenever direct means for system evaluation are absent asin the case of very simple embedded systems (having no display, no filesystem reduced I/O functionalities). Evidently to render flexible and adaptive these capabilities some coding intervention is necessary especially for the interface between DAQ and ROOT (files **DAQSData.cxx** (p. 165) and **DAQPayload.h** (p. 164) in the directory rootbulk).

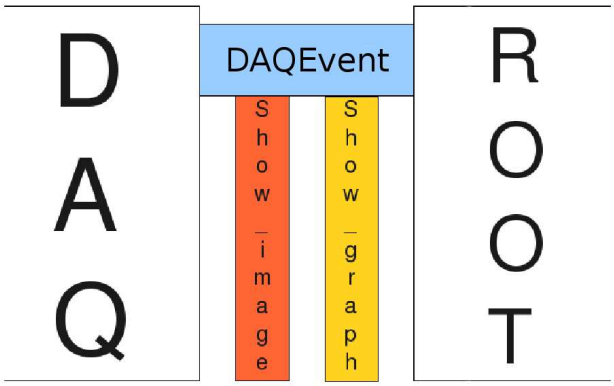


Figure 1.3: Interface DAQ-ROOT

Chapter 2

Usage

2.1 Installation

DAQ platform is possible to compile under Linux and under Windows POSIX emulator Cygwin. To compile correctly DAQ platform is necessary to install:

- Xerces lib (install directly from packet manager: libxerces-c2-dev and libxerces-c28)
- FLTK libraries (install directly from packet manager: libfltk1.1 and libfltk1.1-dev). If is used Cygwin see also **Cygwin issues** (p. 6)
- JPEG libraries (install directly from packet manager: libjeg62 and libjeg62-dev)
- ROOT framework (downlodable from `http://root.cern.ch/drupal/`). If is used Cygwin see also **Cygwin issues** (p. 6).

2.1.1 Cygwin issues

To use FLTK libraries on Cygwin it is needed to fix the jmorecfg.h file, replacing:

- `#ifndef HAVE_BOOLEAN`

  with

  ```
  #if (!defined(HAVE_BOOLEAN)) && (!defined(_RPCNDR_H))
  ```

- `#ifndef XMD_H`

  with

  ```
  #if (!defined(XMD_H) && !defined(_BASETSD_H))
  ```

The only ROOT version working on Cygwin is the 5.11 version, that's included in this package.

Warning:

Remember that ROOT under Cygwin is unsupported.

We provide the DAQ system for Cygwin as a "unsupported" product; We will not guarantee to help you in solving Cygwin/ Windows related bugs/ misbehaviors.

2.2 System personalization

2.2.1 How to add a module?

In this subsection it is explained how to add a new stream class to interface to another communication device. To do this follow these steps:

- Implement the class related to the communication device inheriting from **IStreamDaq** (p. 108) class.
- Update the **Xml-Schema** (p. 9) with the name of the new communication device.
- Update the enumeration of the devices in **daq_enum.h** (p. 154)
- Update the xml parsing procedure
- Update the error enumeration
- Update the **CReader** (p. 61) class

2.2.2 System personalization

In this subsection it is explained how to interface DAQ platform with the embedded system. To do this follow these steps:

- customize the xml file following xml-schema (see **Xml-Schema** (p. 9))
- customize **DAQSData.cxx** (p. 165) and **DAQPayload.h** (p. 164) files to interface DAQ with ROOT repository

Chapter 3

Xml-Schema

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <!-- definition of simple types -->

 <xs:simpleType name="stream_type_style">
  <xs:restriction base="xs:string">
       <xs:enumeration value="serial"/>
   <xs:enumeration value="usb"/>
       <xs:enumeration value="file"/>
       <xs:enumeration value="stdin"/>
       <xs:enumeration value="pipe"/>
   <xs:enumeration value="socket"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:simpleType name="stream_file_type_style">
  <xs:restriction base="xs:string">
   <xs:enumeration value="text"/>
   <xs:enumeration value="bitmap"/>
       <xs:enumeration value="sniffer"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:simpleType name="character_style">
  <xs:restriction base="xs:integer">
   <xs:minInclusive value="0"/>
   <xs:maxInclusive value="255"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:simpleType name="baud_rate_rs232_style">
  <xs:restriction base="xs:positiveInteger">
   <xs:enumeration value="9600"/>
   <xs:enumeration value="19200"/>
   <xs:enumeration value="38400"/>
   <xs:enumeration value="57600"/>
   <xs:enumeration value="115200"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:simpleType name="data_bits_rs232_style">
  <xs:restriction base="xs:positiveInteger">
   <xs:enumeration value="5"/>
   <xs:enumeration value="6"/>
   <xs:enumeration value="7"/>
   <xs:enumeration value="8"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:simpleType name="parity_rs232_style">
  <xs:restriction base="xs:string">
      <xs:enumeration value="odd"/>
   <xs:enumeration value="even"/>
   <xs:enumeration value="none"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:simpleType name="stop_bits_rs232_style">
  <xs:restriction base="xs:float">
   <xs:enumeration value="1"/>
   <xs:enumeration value="1.5"/>
   <xs:enumeration value="2"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:simpleType name="decoder_style">
```

```
 <xs:restriction base="xs:string">
  <xs:enumeration value="xor"/>
  <xs:enumeration value="avr"/>
  <xs:enumeration value="none"/>
 </xs:restriction>
</xs:simpleType>

<xs:simpleType name="h_type_style">
 <xs:restriction base="xs:string">
  <xs:enumeration value="hex"/>
  <xs:enumeration value="char"/>
 </xs:restriction>
</xs:simpleType>

<xs:simpleType name="h_style_style">
 <xs:restriction base="xs:string">
     <xs:enumeration value="standard"/>
  <xs:enumeration value="start"/>
  <xs:enumeration value="none"/>
 </xs:restriction>
</xs:simpleType>

<xs:simpleType name="data_type_style">
 <xs:restriction base="xs:string">
  <xs:enumeration value="int8"/>
  <xs:enumeration value="uint8"/>
  <xs:enumeration value="int16"/>
  <xs:enumeration value="uint16"/>
  <xs:enumeration value="int32"/>
  <xs:enumeration value="uint32"/>
  <xs:enumeration value="int64"/>
  <xs:enumeration value="uint64"/>
  <xs:enumeration value="float32"/>
  <xs:enumeration value="float64"/>
 </xs:restriction>
</xs:simpleType>

<xs:simpleType name="type_style">
 <xs:restriction base="xs:string">
  <xs:enumeration value="image"/>
  <xs:enumeration value="nothing"/>
  <xs:enumeration value="table"/>
  <xs:enumeration value="histogram"/>
 </xs:restriction>
</xs:simpleType>

<!-- definition of attributes -->

<xs:attribute name="title" type="xs:string"/>

<xs:attribute name="stream_type" type="stream_type_style"/>
<xs:attribute name="stream_name" type="xs:string"/>
<xs:attribute name="stream_file_type" type="stream_file_type_style"/>
<xs:attribute name="stream_info" type="xs:nonNegativeInteger"/>

<xs:attribute name="baud_rate" type="baud_rate_rs232_style"/>
<xs:attribute name="data_bits" type="data_bits_rs232_style"/>
<xs:attribute name="parity" type="parity_rs232_style"/>
<xs:attribute name="stop_bits" type="stop_bits_rs232_style"/>

<xs:attribute name="h_style" type="h_style_style"/>

  <xs:attribute name="decoder" type="decoder_style"/>
<xs:attribute name="start_char" type="xs:string"/>
<xs:attribute name="end_char" type="xs:string"/>
<xs:attribute name="esc_char" type="xs:string"/>
<xs:attribute name="s_type_opt" type="h_type_style"/>
```

```
<xs:attribute name="handshake_send" type="xs:string"/>
<xs:attribute name="handshake_ack" type="xs:string"/>
<xs:attribute name="handshake_nack" type="xs:string"/>
<xs:attribute name="h_type_pkt" type="h_type_style"/>

<xs:attribute name="length_pkt" type="xs:positiveInteger"/>

<xs:attribute name="data_name" type="xs:string"/>
<xs:attribute name="data_position" type="xs:nonNegativeInteger"/>
<xs:attribute name="data_size" type="xs:positiveInteger"/>
<xs:attribute name="data_type" type="data_type_style"/>
<xs:attribute name="type" type="type_style"/>

<!-- definition of complex elements -->

<xs:element name="definition">
    <xs:complexType>
  <xs:attribute ref="title" use="required"/>
 </xs:complexType>
  </xs:element>

<xs:element name="serial_stream">
    <xs:complexType>
  <xs:attribute ref="baud_rate"/>
  <xs:attribute ref="data_bits"/>
  <xs:attribute ref="parity"/>
  <xs:attribute ref="stop_bits"/>
 </xs:complexType>
</xs:element>

<xs:element name="stream_opt">
 <xs:complexType>
  <xs:attribute ref="decoder" use="required"/>
  <xs:attribute ref="start_char" use="required"/>
  <xs:attribute ref="end_char" use="required"/>
  <xs:attribute ref="esc_char"/>
  <xs:attribute ref="s_type_opt" use="required"/>
 </xs:complexType>
</xs:element>

<xs:element name="handshake_pkt">
    <xs:complexType>
  <xs:attribute ref="handshake_send" use="required"/>
  <xs:attribute ref="handshake_ack"/>
  <xs:attribute ref="handshake_nack"/>
  <xs:attribute ref="h_type_pkt" use="required"/>
 </xs:complexType>
</xs:element>

<xs:element name="stream">
    <xs:complexType>
  <xs:sequence>
   <xs:element ref="serial_stream" minOccurs="0" maxOccurs="1"/>
   <xs:element ref="stream_opt" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
         <xs:attribute ref="stream_type" use="required"/>
         <xs:attribute ref="stream_name" use="required"/>
         <xs:attribute ref="stream_file_type"/>
         <xs:attribute ref="stream_info"/>
     </xs:complexType>
  </xs:element>

<xs:element name="handshake">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="handshake_pkt" minOccurs="0" maxOccurs="unbounded"/>
```

```
   </xs:sequence>
   <xs:attribute ref="h_style" use="required"/>
  </xs:complexType>
 </xs:element>

 <xs:element name="data">
  <xs:complexType>
   <xs:attribute ref="data_name" use="required"/>
   <xs:attribute ref="data_position" use="required"/>
   <xs:attribute ref="data_size" use="required"/>
   <xs:attribute ref="data_type" use="required"/>
   <xs:attribute ref="type" use="required"/>
  </xs:complexType>
 </xs:element>

 <xs:element name="packet_description">
      <xs:complexType>
         <xs:sequence>
           <xs:element ref="data" minOccurs="1" maxOccurs="unbounded"/>
         </xs:sequence>
         <xs:attribute ref="length_pkt" use="required"/>
      </xs:complexType>
   </xs:element>

   <xs:element name="input_definition">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="definition" minOccurs="1" maxOccurs="1"/>
    <xs:element ref="stream" minOccurs="1" maxOccurs="1"/>
    <xs:element ref="handshake" minOccurs="1" maxOccurs="1"/>
    <xs:element ref="packet_description" minOccurs="1" maxOccurs="1"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

</xs:schema>
```

Chapter 4

EXAMPLE : acquisition of a stream from a socket

DAQ libraries give the possibilities to communicate and parse data from a socket, where the DAQ is the client and the source of the stream is the server. In this case it is implemented a simple server thread (see SocketExample.cpp) that after a simple handshake sends packets shaped like: <DAQ*NNNNNNNN>, where <, > are respectively the start and the end syncronization character, and NNNNNNNN is an UINT32 counter.

4.1 How does SocketExample.cpp work?

This is the implementation of a socket server to communicate with DAQ; the port number is passed as an argument (i.e. ./exe/SocketExample 2000).

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

Checking the correct sintax for SocketExample esecution: port number has to be passed as parameter

```
 if (argc < 2) {
  cout << "Synapsis: ./exe/SocketExample <Port.Number> " <<  endl;
  exit(1);
 }
```

Opening a socket to be used in internet domain using TCP

```
 sockfd = socket(AF_INET, SOCK_STREAM, 0);
 if (sockfd < 0) {
  cerr << "ERROR opening socket" << endl;
   exit(1);
 }
```

Filling the server address structure

```
 bzero((char *) &serv_addr, sizeof(serv_addr));
 portno = atoi(argv[1]);
 serv_addr.sin_family = AF_INET;  //Internet socket
 serv_addr.sin_addr.s_addr = INADDR_ANY; //Connect to any client
 serv_addr.sin_port = htons(portno); //Port number
```

Binding operation (server side only)

```
 if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0){
  cerr << "ERROR on binding" << endl;
  exit(1);
 }
```

Listening for clients connection (blocking function)

```
 listen(sockfd,5);
 clilen = sizeof(cli_addr);
```

Accepting a client and connection

```
 newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, (socklen_t*)&clilen);
 if (newsockfd < 0){
  cerr << "ERROR on accept" << endl;
  exit(1);
 }
```

Handshake:

- Reading the handshaking word

```
bzero(buffer,256);
n = read(newsockfd,buffer,255);
if (n < 0){
 cerr << "ERROR reading from socket" << endl;
 exit(1);
}
```

- Checking if the handshaking packet is correct (it has to be 1) and answer...

```
if(!strcmp (buffer, "1")){
```

... ACK if the packet is correct, ...

```
 n = write(newsockfd, ACK, 1);
 if (n < 0){
  cerr << "ERROR writing to socket" << endl;
  exit(1);
 }
```

...NAK otherwise

```
 n = write(newsockfd, NAK, 1);
 if (n < 0){
  cerr << "ERROR writing to socket" << endl;
  exit(1);
 }
```

Infinite cycle for upgrading the counter and sending packets

```
while(1){
 counter ++;
 unsigned char packet[10];
 packet[0] = 0x02;
 packet[1] = (unsigned char)'D';
 packet[2] = (unsigned char)'A';
 packet[3] = (unsigned char)'Q';
 packet[4] = (unsigned char)'*';
 packet[5] = (unsigned char)((counter >> 24) & 0xFF);
 if((packet[5] == 0x02) || (packet[5] == 0x03))
  packet[5] = 0;
 packet[6] = (unsigned char)((counter >> 16) & 0xFF);
 if((packet[6] == 0x02) || (packet[6] == 0x03))
  packet[6] = 0;
 packet[7] = (unsigned char)((counter >> 8) & 0xFF);
 if((packet[7] == 0x02) || (packet[7] == 0x03))
  packet[7] = 0;
 packet[8] = (unsigned char)((counter) & 0xFF);
 if((packet[8] == 0x02) || (packet[8] == 0x03))
  packet[8] = 0;
 packet[9] = 0x03;
 n = write(newsockfd, packet, 10);
 if (n < 0){
  cerr << "ERROR writing to socket" << endl;
  exit(1);
 }else
 cout << "Sent packet: " << hex << (int)packet[0] << hex << (int)packet[1] << hex << (int)packet[2] << hex << (int)pack
  hex <<(int) packet[4] << hex << (int)packet[5] << hex << (int)packet[6] << hex << (int)packet[7] << hex << (int)packe
  hex << (int)packet[9] << hex <<(int) packet[10] << endl;
}


return 0;
```

4.2 XML file example for socket acquisition

This file.xml fixes the socket settings, the handshake word to initialize the communication, the packet lenght and the data topology in a packet.

4.2.1 Stream title (REQUIRED)

```
<definition title="DAQ: Simple socket acquisition"/>
```

4.2.2 Stream definition (REQUIRED)

- stream_type: type of stream (the allowed value are file, serial, pipe, socket). REQUIRED
- stream_name: the name or the address of the file to be written (/dev/ttyUSB0 for serial, or IP-address or 'localhost' for socket, or the name of the file, or the pipe name. REQUIRED.
- stream_info: more info about the stream (i.e. the port number for the socket). REQUIRED
  In this case DAQ will use a socket that it has to connect to the IP-address 'localhost' through the port number 2000.

  ```
  <stream stream_type="socket" stream_name="localhost" stream_info="2000">
  ```

- stream option (NOT REQUIRED):

  1. decoder: used decoder type (allowable value are 'none' and 'xor')

  2. esc_char: escape character (for decoder). Could be expressed like a character or like an hexadecimal number (see s_type_opt)

  3. start_char: syncronization start char. Could be expressed like a character or like an hexadecimal number (see s_type_opt)

  4. end_char: syncronization end char. Could be expressed like a character or like an hexadecimal number (see s_type_opt)

  5. s_type_opt: the way to interpret the selected character:

  ..a. 'char' all the character are interpreted like character;

  ..b. 'hex' all the character are interpreted like hexadecimal value.

  In this case the start character will be 0x02 and the end character will be 0x03.

  ```
  <stream_opt decoder="none" start_char="02" end_char="03" s_type_opt="hex"/>
  ```

4.2.3 Handshake definition (REQUIRED)

The attribute of the node handshake explain wich type of handshake is choosen:

- 'none': no handshake
- 'start': handshake is done to syncronize DAQ with the stream and/or to set-up the device.
- 'standard': the packet to be parsed by DAQ is an answer to the unique handshake packet.

Every handshake packet is describe by this field: (NOT REQUIRED)

- handshake_send: packet to send (REQUIRED in case of 'start' and 'standard')

- handshake_ack: acknowledgement to the packet (only in case of 'start')
- handshake_nack: not acknowledgement to the packet (only in case of 'start')
- h_type_pkt: the way to interpret the selected character:
  1. 'char' all the character are interpreted like character;
  2. 'hex' all the character are interpreted like hexadecimal value.

In this case the handshake type is 'start', the packet to send is '1' (0x31), ACK packet is '1' (0x31) and NAK packet '0' (0x30).

```
<handshake h_style="start">
 <handshake_pkt handshake_send="31" handshake_ack="31" handshake_nack="30" h_type_pkt="hex"/>
</handshake>
```

4.2.4 Payload definition

In this part is described how the packet to parse are organized:

- length_pkt : total packet length in bytes (REQUIRED)
- n data node as the number of the fields in the packet. Every node is composed with this attributes:

  ..1) data_name: name of the field (REQUIRED)

  ..2) data_position: absolute position from the start of the packet in number of bytes (REQUIRED)

  ..3) data_size: size of the field in number of bytes (REQUIRED)

  ..4) data_type: data type (INT8, UINT8, INT16, UINT16, INT32, UINT32, INT64, UINT64, FLOAT32, FLOAT64) (REQUIRED)

  ..5) type: semantic of the data (DS_NOTHING, DS_TABLE, DS_IMAGE, DS_HISTOGRAM (REQUIRED))

```
<packet_description length_pkt="8">
 <data data_name="Message" data_position="0" data_size="3" data_type="int8" type="table"/>
 <data data_name="Divisor" data_position="3" data_size="1" data_type="uint8" type="nothing"/>
 <data data_name="Counter" data_position="4" data_size="4" data_type="uint32" type="table"/>
</packet_description>
```

Warning:

These five character entities are assumed to be predeclared, and it is possible to use without declaring them:

- "<": the less-than character (<) starts element markup (the first character of a start-tag or an end-tag).
- "&": the ampersand character (&) starts entity markup (the first character of a character entity reference).
- ">": the greater-than character (>) ends a start-tag or an end-tag.
- """: the double-quote character (") can be symbolised with this character entity reference when you need to embed a double-quote inside a string which is already double-quoted.
- "'": the apostrophe or single-quote character (') can be symbolised with this character entity reference when you need to embed a single-quote or apostrophe inside a string which is already single-quoted.

4.2.5 socketLinux.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<input_definition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="input.xsd">
 <definition title="DAQ: Simple socket acquisition"/>

 <stream stream_type="socket" stream_name="localhost" stream_info="2000">
  <stream_opt decoder="none" start_char="02" end_char="03" s_type_opt="hex"/>
 </stream>

 <handshake h_style="start">
  <handshake_pkt handshake_send="31" handshake_ack="31" handshake_nack="30" h_type_pkt="hex"/>
 </handshake>

 <packet_description length_pkt="8">
  <data data_name="Message" data_position="0" data_size="3" data_type="int8" type="table"/>
  <data data_name="Divisor" data_position="3" data_size="1" data_type="uint8" type="nothing"/>
  <data data_name="Counter" data_position="4" data_size="4" data_type="uint32" type="table"/>
 </packet_description>
</input_definition>
```

4.3 Payload analysis example

In this case the **DAQEvent** (p. 92) class is projected to store data from the socket, and in the main(...) function (in DAQ.cpp) the event->Build(...) function is called every packet received.

```
  event->Build(ev++, data_formatted);
tree->Fill();  //fill the tree
*
```

4.4 Post-processing socket stream

The file LoopDAQEvent.cpp is the application to process data stored inside ROOT repository (default name is **test.root**). The logic of this simple programm is:

- read the element

```
TClonesArray *myarray = event->GetDAQSDatas();
     if (myarray) mydaq = (DAQSData *) myarray->First();
```

- display data

```
  cout << "GetMessage: " << mydaq->getMessage() << endl;
cout << "GetCounter: " << mydaq->getCounter() << endl;
```

- process the counter

```
  nselected++;
count += (Double_t)mydaq->getCounter() / (nevent - nselected);
```

Chapter 5

EXAMPLE : serial acquisition of a picture from HV7131GP and FLEX

The firmware is downlodable from `http://www.evidence.eu.com/content/view/329/266/`.

5.1 XML file example for image acquisition via RS-232

This file.xml fixes the RS-232 settings, the handshake word to initialize the camera, the packet lenght and the data topology in a packet.

5.1.1 Stream title (REQUIRED)

```
<definition title="DAQ: Simple serial acquisition"/>
```

5.1.2 Stream definition (REQUIRED)

- stream_type: type of stream (the allowed value are file, serial, pipe, socket). REQUIRED
- stream_name: the name or the address of the file to be written (/dev/ttyUSB0 for serial, or IP-address or 'localhost' for socket, or the name of the file, or the pipe name. (REQUIRED)
- stream_info: more info about the stream (i.e. the port number for the socket). REQUIRED

  ```
  <stream stream_type="serial" stream_name="/dev/ttyUSB0" stream_file_type="text" stream_info="1">
  ```

- serial preset (REQUIRED only for SERIAL):

  1. baud_rate: baud rate for serial communication (to check the allowable values see input.xsd schema file)
  2. data_bits: length in bits of a character received (to check the allowable values see input.xsd schema file)
  3. parity: parity bit information
  4. stop_bits : number of stop bits

  The serial stream selected will have baud-rate 115200, 8 bits for character, no parity bit and 1 stop bit.

  ```
  <serial_stream baud_rate="115200" data_bits="8" parity="none" stop_bits="1"/>
  ```

5.1.3 Handshake definition

The attribute of the node handshake explain wich type of handshake is choosen:

- 'none': no handshake
- 'start': handshake is done to syncronize DAQ with the stream and/or to set-up the device.
- 'standard': the packet to be parsed by DAQ is an answer to the unique handshake packet.

Every handshake packet is describe by this field:

- handshake_send: packet to send (REQUIRED in case of 'start' and 'standard')
- handshake_ack: acknowledgement to the packet (only in case of 'start')

- handshake_nack: not acknowledgement to the packet (only in case of 'start')
- h_type_pkt: the way to interpret the selected character:
  1. 'char' all the character are interpreted like character;
  2. 'hex' all the character are interpreted like hexadecimal value.

In this case the handshake type is 'start', the packets to send are:

- '1' (0x31)
- '*01*01*' (0x2A 0x30 0x31 0x2A 0x30 0x31 0x2A)

ACK packets are:

- '1' (0x31)
- '1' (0x31)

and NAK packets are:

- '0' (0x30)
- '0' (0x30)

```
<handshake h_style="start">
 <handshake_pkt handshake_send="31" handshake_ack="31" handshake_nack="30" h_type_pkt="hex"/>
 <handshake_pkt handshake_send="2A30312A30312A" handshake_ack="31" handshake_nack="30" h_type_pkt="hex"/>
</handshake>
```

5.1.4 Payload definition

In this part is described how the packet to parse are organized:

- length_pkt : total packet length in bytes (REQUIRED)
- n data node as the number of the fields in the packet. Every node is composed with this attributes:

  ..1) data_name: name of the field

  ..2) data_position: absolute position from the start of the packet in number of bytes

  ..3) data_size: size of the field in number of bytes

  ..4) data_type: data type (INT8, UINT8, INT16, UINT16, INT32, UINT32, INT64, UINT64, FLOAT32, FLOAT64)

  ..5) type: semantic of the data (DS_NOTHING, DS_TABLE, DS_IMAGE, DS_HISTOGRAM)

```
<packet_description length_pkt="19215">
 <data data_name="Header" data_position="0" data_size="3" data_type="uint8" type="nothing"/>
 <data data_name="Width" data_position="3" data_size="2" data_type="uint16" type="table"/>
 <data data_name="Height" data_position="5" data_size="2" data_type="uint16" type="table"/>
 <data data_name="Duration" data_position="7" data_size="4" data_type="uint32" type="table"/>
 <data data_name="Lost" data_position="11" data_size="2" data_type="uint16" type="table"/>
 <data data_name="YMean" data_position="13" data_size="2" data_type="uint16" type="table"/>
 <data data_name="Image" data_position="15" data_size="19200" data_type="uint8" type="image"/>
</packet_description>
```

Warning:

These five character entities are assumed to be predeclared, and it is possible to use without declaring them:

- "<": the less-than character (<) starts element markup (the first character of a start-tag or an end-tag).
- "&": the ampersand character (&) starts entity markup (the first character of a character entity reference).
- ">": the greater-than character (>) ends a start-tag or an end-tag.
- """: the double-quote character (") can be symbolised with this character entity reference when you need to embed a double-quote inside a string which is already double-quoted.
- "'": the apostrophe or single-quote character (') can be symbolised with this character entity reference when you need to embed a single-quote or apostrophe inside a string which is already single-quoted.

5.1.5 serialImageLinux.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<input_definition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="input.xsd">
 <definition title="DAQ: Simple serial acquisition"/>

 <stream stream_type="serial" stream_name="/dev/ttyUSB0" stream_file_type="text" stream_info="1">
  <serial_stream baud_rate="115200" data_bits="8" parity="none" stop_bits="1"/>
 </stream>

 <handshake h_style="start">
  <handshake_pkt handshake_send="31" handshake_ack="31" handshake_nack="30" h_type_pkt="hex"/>
  <handshake_pkt handshake_send="2A30312A30312A" handshake_ack="31" handshake_nack="30" h_type_pkt="hex"/>
 </handshake>

 <packet_description length_pkt="19215">
  <data data_name="Header" data_position="0" data_size="3" data_type="uint8" type="nothing"/>
  <data data_name="Width" data_position="3" data_size="2" data_type="uint16" type="table"/>
  <data data_name="Height" data_position="5" data_size="2" data_type="uint16" type="table"/>
  <data data_name="Duration" data_position="7" data_size="4" data_type="uint32" type="table"/>
  <data data_name="Lost" data_position="11" data_size="2" data_type="uint16" type="table"/>
  <data data_name="YMean" data_position="13" data_size="2" data_type="uint16" type="table"/>
  <data data_name="Image" data_position="15" data_size="19200" data_type="uint8" type="image"/>
 </packet_description>
</input_definition>
```

5.2 Payload analysis example

In this case the **DAQEvent** (p. 92) class is projected to store an image of 19200 bytes (160x120 pixel), and in the main(...) function (in DAQ.cpp) the event->Build(...) function is called every packet received.

```
  event->Build(ev++, data_formatted);
tree->Fill();  //fill the tree
*
```

5.3 Post-processing images

The file LoopDAQEvent.cpp is the application to process the images stored inside ROOT repository (default name is **test.root**). The paradigm of this simple programm is:

- read the element

```
TClonesArray *myarray = event->GetDAQSDatas();
     if (myarray) mydaq = (DAQSData *) myarray->First();
```

- calculate the likely-hood value using FDY and filling histogram

```
my[graph_abscissa] = fdy_cd(mydaq->getSImage(), 19200);
      TAxis *ax = hp[1]->GetXaxis();
     hp[1]->SetBinContent((ax->GetBinCenter((Float_t)graph_abscissa)) + 2, y[graph_abscissa]);
     graph_abscissa += 1;
```

- show stored image

```
mydaq->DAQSDataInit();
    mydaq->DAQSDataRender();
```

Chapter 6

Todo List

Member CFile::retStream (p. **43**)**(unsigned char **buffer, unsigned long *length)**
Introduce other feof check and ferror controls

Member CFile::File_handshaking (p. **44**)**(handshake_style style)** Handshaking for file

Member CReader::getDefinition (p. **62**)**(struct SDef** (p. **116**) **def)** usb

Member CReader::getDefinition (p. **62**)**(struct SDef** (p. **116**) **def)** stdin

Chapter 7

Class Index

7.1 Class Hierarchy

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

Chapter 8

Class Index

8.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 9

File Index

9.1 File List

Here is a list of all files with brief descriptions:

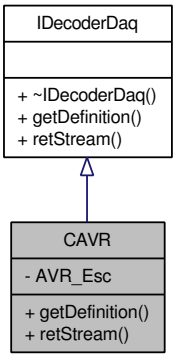
Chapter 10

Class Documentation

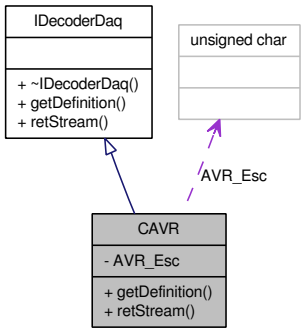
10.1 CAVR Class Reference

Implementation of **IDecoderDaq** (p. 102) interface to decode 0x0D from AVR RS-232.

Inheritance diagram for CAVR:



Collaboration diagram for CAVR:



Public Member Functions

- virtual **s_ret getDefinition** (struct **SDef** def)

- virtual **s_ret retStream** (unsigned char *buffer_i, unsigned long length_i, unsigned char **buffer_o, unsigned long *length_o)

Private Attributes

- unsigned char **AVR_Esc**

10.1.1 Detailed Description

Warning:

The character 0x0A coming from RS-232 communication from AVR microcontroller is translated in 0x0D 0x0A

10.1.2 Member Function Documentation

10.1.2.1 s_ret CAVR::getDefinition (struct SDef *def*) [virtual]

Initialize the stream decoder.

Parameters:

def Pointer to a stream definition structure (input)

Returns:

This method return one of the possible value from s_ret

See also:

s_ret (p. 196)

Implements **IDecoderDaq** (p. 102).

10.1.2.2 s_ret CAVR::retStream (unsigned char * *buffer_i*, unsigned long *length_i*, unsigned char ** *buffer_o*, unsigned long * *length_o*) [virtual]

Method that return the decoded packet.

Parameters:

buffer_i Pointer of the buffet to decode (input)
length_i Length of the buffer to decode (input)
buffer_o Pointer to a pointer of a vector of byte to be formatted (output)
length_o Pointer to the packet lenght variable (output)

Returns:

This method return one of the possible value from s_ret

See also:

s_ret (p. 196)

Implements **IDecoderDaq** (p. 103).

10.1.3 Member Data Documentation

10.1.3.1 unsigned char CAVR::AVR_Esc [private]

Escape character

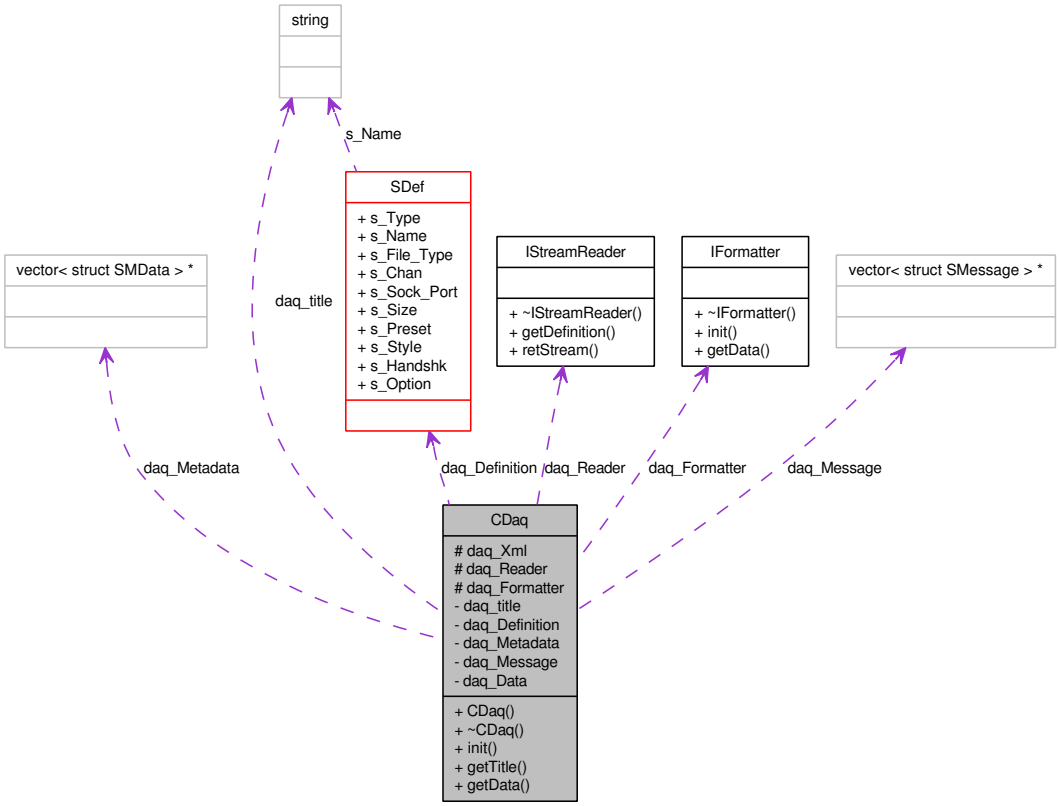
The documentation for this class was generated from the following files:

- **CAVR.h**
- **CAVR.cpp**

10.2 CDaq Class Reference

User interface from the data acquisition library and the user application.

Collaboration diagram for CDaq:



Public Member Functions

- **CDaq** (char *xmlFile)
- string **getTitle** (void)
- **daq_ret getData** (vector< vector< **SData** > > *data)

Protected Attributes

- **IXml** * **daq_Xml**
- **IStreamReader** * **daq_Reader**
- **IFormatter** * **daq_Formatter**

Private Attributes

- string **daq_title**
- struct **SDef** * **daq_Definition**
- vector< struct **SMData** > * **daq_Metadata**
- vector< struct **SMessage** > * **daq_Message**
- vector< **SData** > * **daq_Data**

10.2.1 Constructor & Destructor Documentation

10.2.1.1 CDaq::CDaq (char * *xmlFile*)

CDaq (p. 38) constructor.

Parameters:

xmlFile pointer to the string of the xml file path

10.2.1.2 CDaq::~CDaq ()

CDaq (p. 38) destructor.

10.2.2 Member Function Documentation

10.2.2.1 daq_ret CDaq::init (void)

Init method to initialize the background classes with the data parsed from the xml file.

< Value to be added to CXmlParser return value to be translated to **CDaq** (p. 38) return value

See also:

daq_ret (p. 158)
x_ret (p. 199)

< Value to be added to CXmlParser return value to be translated to **CDaq** (p. 38) return value

See also:

daq_ret (p. 158)
x_ret (p. 199)

< Value to be added to **CReader** (p. 61) return value to be translated to **CDaq** (p. 38) return value

See also:

daq_ret (p. 158)
s_ret (p. 196)

< Value to be added to **CFormatter** (p. 46) return value to be translated to **CDaq** (p. 38) return value

See also:

daq_ret (p. 158)
f_ret (p. 175)

10.2.2.2 string CDaq::getTitle (void)

Method that returns the title of the title.

Returns:

Returns the string of the title

10.2.2.3 daq_ret CDaq::getData (vector< vector< SData > > * *data*)

Method that formats data and create the vector of data formatted.

Parameters:

buffer Pointer to a vector that contain the n formatted data vector related to the n messages received from the reader(output)

Returns:

Returns the string of the title

< Value to be added to **CReader** (p. 61) return value to be translated to **CDaq** (p. 38) return value

See also:

daq_ret (p. 158)
s_ret (p. 196)

< Value to be added to **CFormatter** (p. 46) return value to be translated to **CDaq** (p. 38) return value

See also:

daq_ret (p. 158)
f_ret (p. 175)

10.2.3 Member Data Documentation

10.2.3.1 IXml* CDaq::daq_Xml [protected]

Instance of the xml parser interface.

10.2.3.2 IStreamReader* CDaq::daq_Reader [protected]

Instance of data reader interface.

10.2.3.3 IFormatter* CDaq::daq_Formatter [protected]

Instance of the formatter interface

10.2.3.4 string CDaq::daq_title [private]

Variable that contains the title

10.2.3.5 struct SDef* CDaq::daq_Definition [read, private]

Varibale of **SDef** (p. 116) struct that define the stream

See also:

SDef (p. 116)

10.2.3.6 vector<struct SMData>* CDaq::daq_Metadata [private]

Vector of metadata

See also:

SMData (p. 121)

10.2.3.7 vector<struct SMessage>* CDaq::daq_Message [private]

Vector of the messages (output of the reader)

10.2.3.8 vector<SData>* CDaq::daq_Data [private]

Formatted data vector

See also:

SData (p. 114)

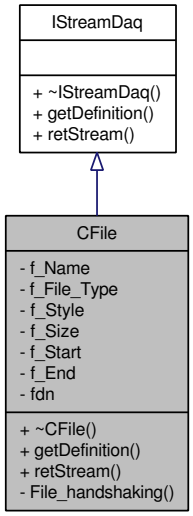
The documentation for this class was generated from the following files:

- **CDaq.h**
- **CDaq.cpp**

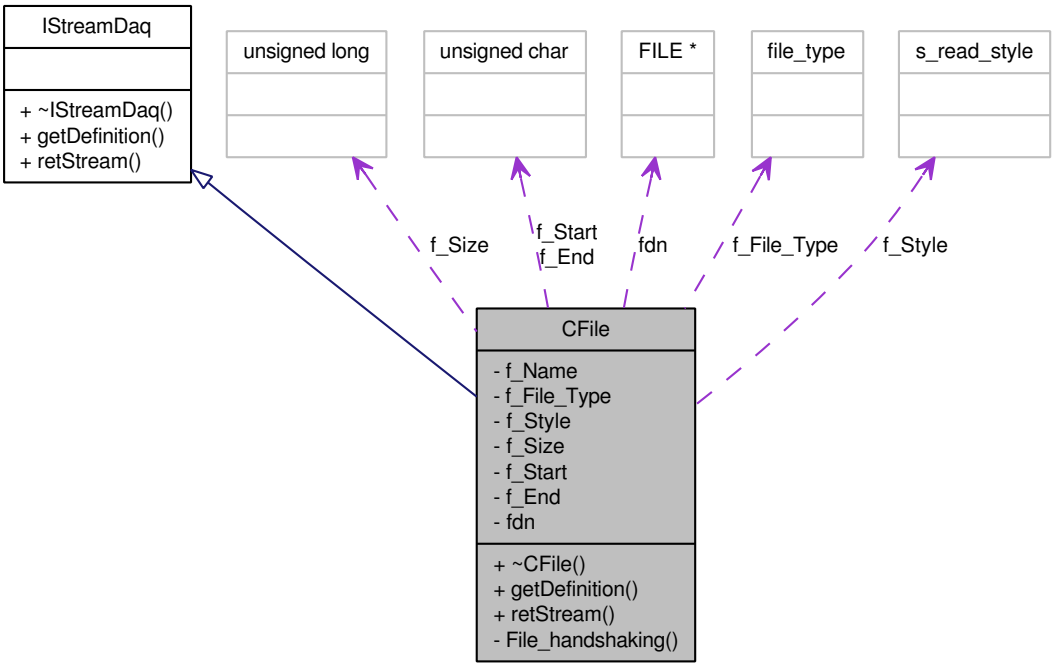
10.3 CFile Class Reference

Implementation of IStream interface to read a file. This class reads a file and creates packet shaped like the **SDef** (p. 116) structure directives.

Inheritance diagram for CFile:



Collaboration diagram for CFile:



Public Member Functions

- virtual **s_ret getDefinition** (struct **SDef** def, **s_read_style** sty)

- virtual **s_ret retStream** (unsigned char **buffer, unsigned long *length)

Private Member Functions

- **s_ret File_handshaking** (**handshake_style** style)

Private Attributes

- char * **f_Name**
- **file_type f_File_Type**
- **s_read_style f_Style**
- unsigned long **f_Size**
- unsigned char **f_Start**
- unsigned char **f_End**
- FILE * **fdn**

10.3.1 Constructor & Destructor Documentation

10.3.1.1 CFile::~CFile ()

CFile (p. 42) destructor.

10.3.2 Member Function Documentation

10.3.2.1 s_ret CFile::getDefinition (struct SDef *def*, s_read_style *sty*) [virtual]

Initialize the stream reader (.

Parameters:

def Pointer to a stream definition structure (input)

sty Style of reading (by length, or by start and end character)

Returns:

This method return one of the possible value from s_ret

See also:

s_ret (p. 196)

Implements **IStreamDaq** (p. 109).

10.3.2.2 s_ret CFile::retStream (unsigned char ** *buffer*, unsigned long * *length*) [virtual]

Blocking method to wait the data to be passed to formatter (maybe after a decode and a "divide in message" processes respectively to interpret the escape charactera and to divide packets to the correct length).

Parameters:

buffer Pointer to a pointer of a vector of byte to be formatted (output)

length Pointer to the packet lenght variable (output)

Returns:

This method return one of the possible value from s_ret

See also:

s_ret (p. 196)

Todo

Introduce other feof check and ferror controls

Implements **IStreamDaq** (p. 109).

10.3.2.3 s_ret CFile::File_handshaking (handshake_style *style*) [private]

This function execute the handshaking to initialize the stream.

Parameters:

style Handshacking style

Returns:

s_ret value about sending from

See also:

s_ret (p. 196)

Todo

Handshaking for file

10.3.3 Member Data Documentation

10.3.3.1 char* CFile::f_Name [private]

File path

10.3.3.2 file_type CFile::f_File_Type [private]

File type

10.3.3.3 s_read_style CFile::f_Style [private]

Reading style

10.3.3.4 unsigned long CFile::f_Size [private]

Packet total size

10.3.3.5 unsigned char CFile::f_Start [private]

Start character

10.3.3.6 unsigned char CFile::f_End [private]

End character

10.3.3.7 FILE* CFile::fdn [private]

File descriptor number

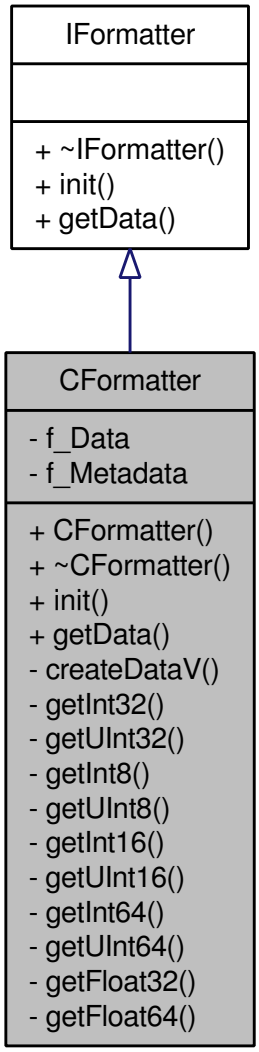
The documentation for this class was generated from the following files:

- **CFile.h**
- **CFile.cpp**

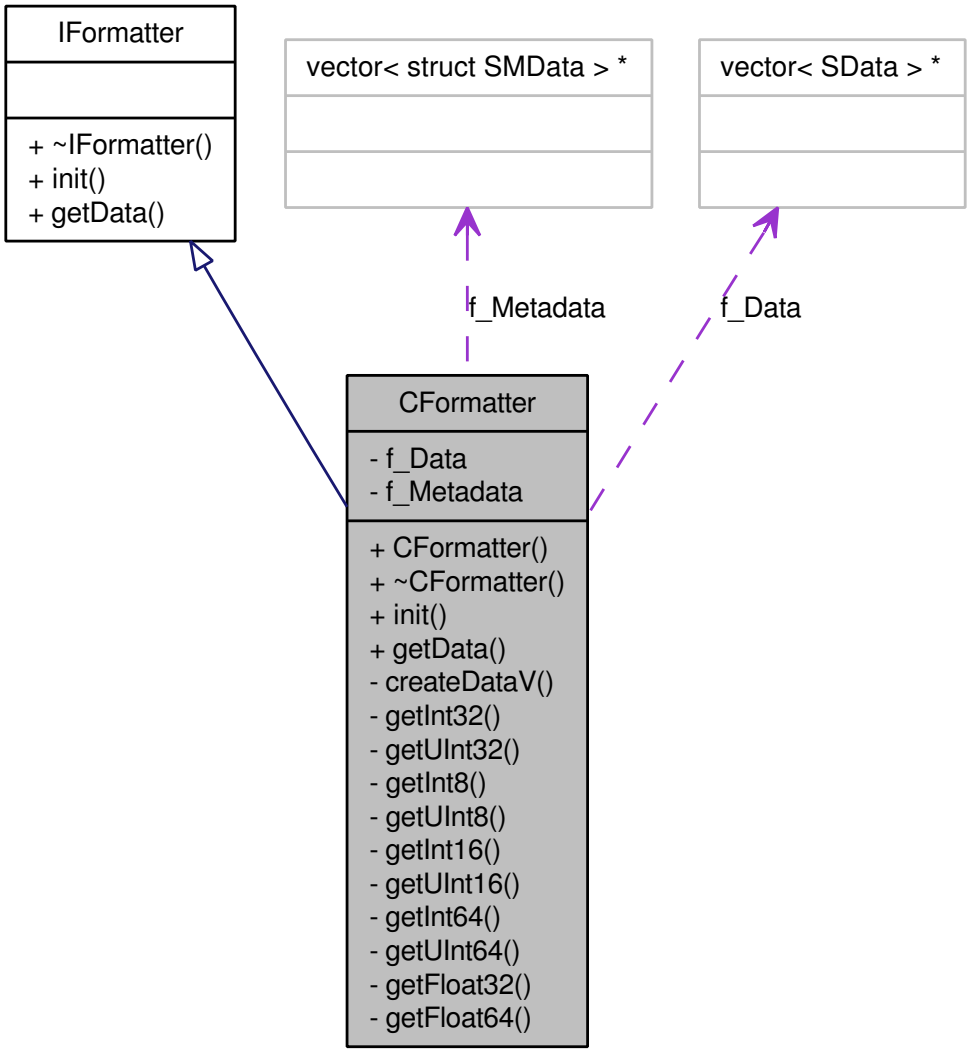
10.4 CFormatter Class Reference

Class to format data received from stream based on the metadata vector (created by the xml parser).

Inheritance diagram for CFormatter:



Collaboration diagram for CFormatter:



Public Member Functions

- **CFormatter** (vector< **SData** > *data)
- virtual **f_ret init** (vector< struct **SMData** > *metadata)
- virtual **f_ret getData** (unsigned char *pkt)

Private Member Functions

- **f_ret createDataV** ()
- int * **getInt32** (unsigned char *v, unsigned long _size)
- unsigned int * **getUInt32** (unsigned char *v, unsigned long _size)
- char * **getInt8** (unsigned char *v, unsigned long _size)
- unsigned char * **getUInt8** (unsigned char *v, unsigned long _size)
- short * **getInt16** (unsigned char *v, unsigned long _size)

- unsigned short * **getUInt16** (unsigned char *v, unsigned long _size)
- long * **getInt64** (unsigned char *v, unsigned long _size)
- unsigned long * **getUInt64** (unsigned char *v, unsigned long _size)
- float * **getFloat32** (unsigned char *v, unsigned long _size)
- double * **getFloat64** (unsigned char *v, unsigned long _size)

Private Attributes

- vector< **SData** > * **f_Data**
- vector< struct **SMData** > * **f_Metadata**

10.4.1 Detailed Description

See also:

SMData (p. 121)

10.4.2 Constructor & Destructor Documentation

10.4.2.1 CFormatter::CFormatter (vector< SData > * *data*)

Constructor.

Parameters:

header Pointer to header string vector

data Pointer to formatted data vector

10.4.2.2 CFormatter::~CFormatter ()

Destructor.

10.4.3 Member Function Documentation

10.4.3.1 f_ret CFormatter::init (vector< struct SMData > * *metadata*) [virtual]

This method creates the header vector and the formatted data vector.

Parameters:

metadata Metadata vector

Returns:

One of the possible value of f_ret

See also:

f_ret (p. 175)

Implements **IFormatter** (p. 105).

10.4.3.2 f_ret CFormatter::getData (unsigned char * *pkt*) [virtual]

Fill the vector of formatted data.

Parameters:

msg Pointer to a char vector of data to be formatted received from **IStreamReader** (p. 110) (input)

Returns:

One of the possible value of f_ret

See also:

f_ret (p. 175)

Implements **IFormatter** (p. 105).

10.4.3.3 f_ret CFormatter::createDataV () [private]

This method creates the data structure for the data vector, read by the metadata vector.

Returns:

One of the possible value of f_ret

See also:

f_ret (p. 175)

10.4.3.4 int * CFormatter::getInt32 (unsigned char * *v*, unsigned long *_size*) [private]

Method to format integer values.

Parameters:

v Pointer to a subset of stream data (input)

_size Size of the resulting integer vector (input)

Returns:

This method returns the pointer of the integer vector

10.4.3.5 unsigned int * CFormatter::getUInt32 (unsigned char * *v*, unsigned long *_size*) [private]

Method to format unsigned integer values.

Parameters:

v Pointer to a subset of stream data (input)

_size Size of the resulting unsigned integer vector (input)

Returns:

This method returns the pointer of the unsigned integer vector

10.4.3.6 char * CFormatter::getInt8 (unsigned char * *v*, unsigned long *_size*) [private]

Method to format char values.

Parameters:

v Pointer to a subset of stream data (input)

_size Size of the resulting char vector (input)

Returns:

This method returns the pointer of the char vector

10.4.3.7 unsigned char * CFormatter::getUInt8 (unsigned char * *v*, unsigned long *_size*) [private]

Method to format unsigned char values.

Parameters:

v Pointer to a subset of stream data (input)

_size Size of the resulting unsigned char vector (input)

Returns:

This method returns the pointer of the unsigned char vector

10.4.3.8 short * CFormatter::getInt16 (unsigned char * *v*, unsigned long *_size*) [private]

Method to format short values.

Parameters:

v Pointer to a subset of stream data (input)

_size Size of the resulting short vector (input)

Returns:

This method returns the pointer of the char vector

10.4.3.9 unsigned short * CFormatter::getUInt16 (unsigned char * *v*, unsigned long _ *size*) [private]

Method to format unsigned short values.

Parameters:

v Pointer to a subset of stream data (input)

_size Size of the resulting unsigned short vector (input)

Returns:

This method returns the pointer of the unsigned char vector

10.4.3.10 long * CFormatter::getInt64 (unsigned char * *v*, unsigned long _ *size*) [private]

Method to format long values.

Parameters:

v Pointer to a subset of stream data (input)

_size Size of the resulting long vector (input)

Returns:

This method returns the pointer of the long vector

10.4.3.11 unsigned long * CFormatter::getUInt64 (unsigned char * *v*, unsigned long _ *size*) [private]

Method to format unsigned long values.

Parameters:

v Pointer to a subset of stream data (input)

_size Size of the resulting unsigned long vector (input)

Returns:

This method returns the pointer of the unsigned long vector

10.4.3.12 float * CFormatter::getFloat32 (unsigned char * *v*, unsigned long _ *size*) [private]

Method to format float values.

Parameters:

v Pointer to a subset of stream data (input)

_size Size of the resulting float vector (input)

Returns:

This method returns the pointer of the float vector

10.4.3.13 double * CFormatter::getFloat64 (unsigned char * *v*, unsigned long *_size*) [private]

Method to format double values.

Parameters:

v Pointer to a subset of stream data (input)

_size Size of the resulting double vector (input)

Returns:

This method returns the pointer of the double::include <stdio.h> vector

10.4.4 Member Data Documentation

10.4.4.1 vector<SData>* CFormatter::f_Data [private]

SData (p. 114) vector

See also:

SData (p. 114)

10.4.4.2 vector<struct SMData>* CFormatter::f_Metadata [private]

Metadata vector

See also:

SMData (p. 121)

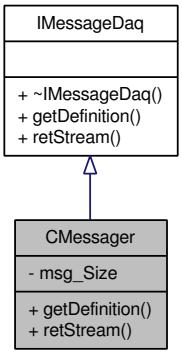
The documentation for this class was generated from the following files:

- **CFormatter.h**
- **CFormatter.cpp**

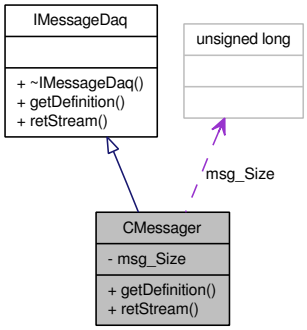
10.5 CMessager Class Reference

Implementation of **IMessageDaq** (p. 106) interface to devide in messages a packet.

Inheritance diagram for CMessager:



Collaboration diagram for CMessager:



Public Member Functions

- virtual **s_ret getDefinition** (struct **SDef** def)
- virtual **s_ret retStream** (unsigned char *buffer_i, unsigned long length_i, std::vector< struct **SMessage** > *buffer_o)

Private Attributes

- unsigned long **msg_Size**

10.5.1 Member Function Documentation

10.5.1.1 s_ret CMessager::getDefinition (struct SDef *def*) [virtual]

Initialize the messages maker.

Parameters:

def Pointer to a stream definition structure (input)

Returns:

This method return one of the possible value from s_ret

See also:

s_ret (p. 196)

Implements **IMessageDaq** (p. 106).

10.5.1.2 s_ret CMessager::retStream (unsigned char * *buffer_i*, unsigned long *length_i*, std::vector< struct SMessage > * *buffer_o*) [virtual]

This method divides the packet recived into message to be formatted (the lenght of each message is the pkt_lenght defined in the xml file).

Parameters:

buffer_i Pointer of the buffet to decode (input)

length_i Length of the buffer to decode (input)

buffer_o Pointer to a vector of the messages to be formatted (output)

Returns:

This method return one of the possible value from s_ret

See also:

s_ret (p. 196)

Implements **IMessageDaq** (p. 107).

10.5.2 Member Data Documentation

10.5.2.1 unsigned long CMessager::msg_Size [private]

Size of a single message

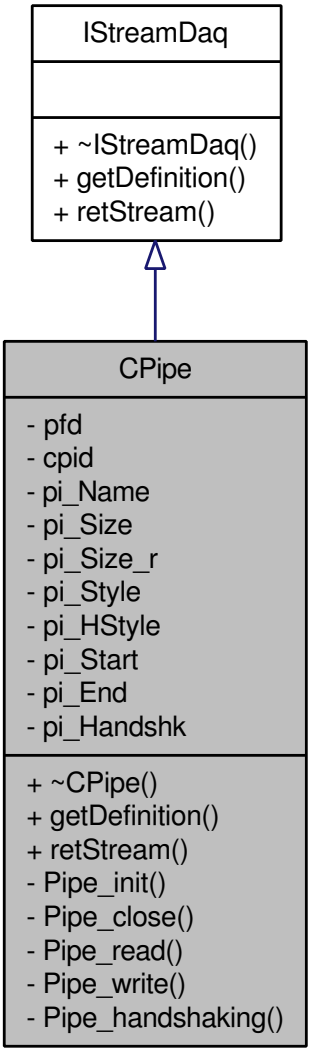
The documentation for this class was generated from the following files:

- **CMessager.h**
- **CMessager.cpp**

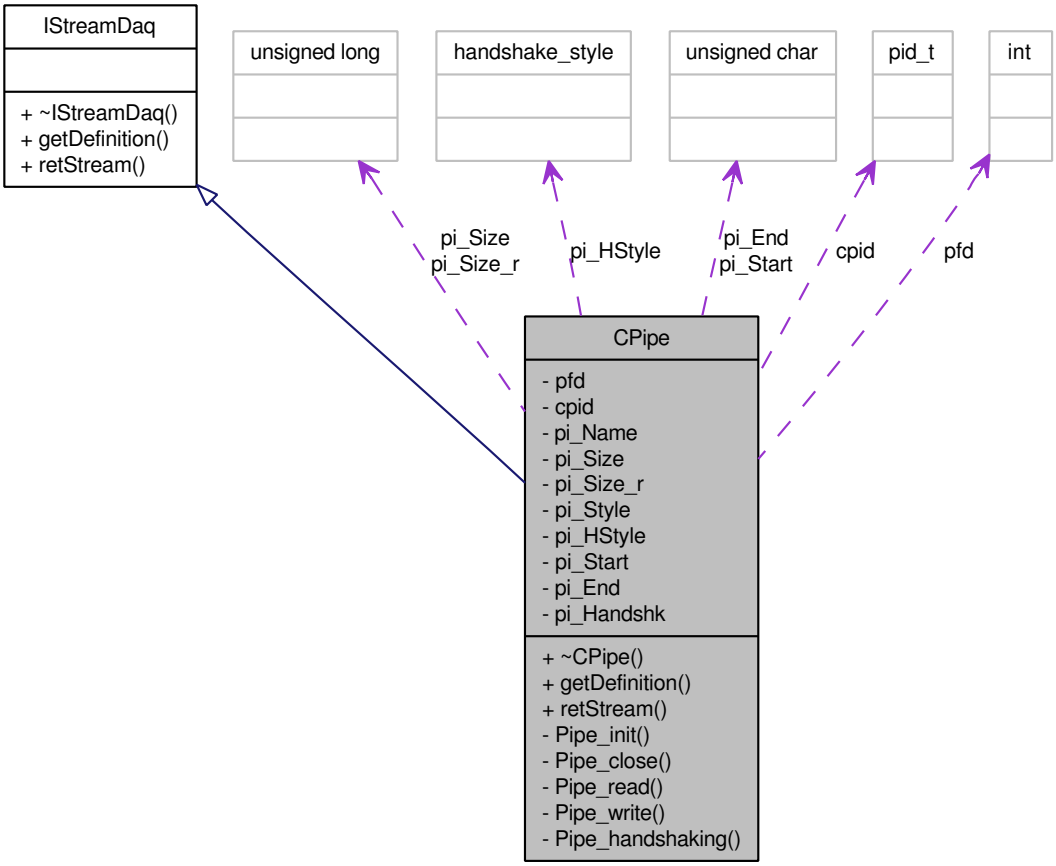
10.6 CPipe Class Reference

Class to read from a pipe (tread intercommunication method). Implementation of IStream interface to read a stream from a pipe. This class reads bytes from a pipe and creates packets shaped like the **SDef** (p. 116) structure directives. Daq is client of the pipe created by the server thread.

Inheritance diagram for CPipe:



Collaboration diagram for CPipe:



Public Member Functions

- virtual **s_ret getDefinition** (struct **SDef** def, **s_read_style** sty)
- virtual **s_ret retStream** (unsigned char **buffer, unsigned long *length)

Private Member Functions

- **s_ret Pipe_init** (void)
- **s_ret Pipe_read** (unsigned char *data, unsigned int data_size)
- **s_ret Pipe_write** (unsigned char *data, unsigned int data_size)
- **s_ret Pipe_handshaking** (**handshake_style** style)

Private Attributes

- int **pfd** [2]
- pid_t **cpid**
- char * **pi_Name**
- unsigned long **pi_Size**
- unsigned long **pi_Size_r**
- **s_read_style pi_Style**

- **handshake_style pi_HStyle**
- unsigned char **pi_Start**
- unsigned char **pi_End**
- std::vector< struct **SHandshaking** > * **pi_Handshk**

10.6.1 Constructor & Destructor Documentation

10.6.1.1 CPipe::~CPipe ()

CPipe (p. 55) destructor.

10.6.2 Member Function Documentation

10.6.2.1 s_ret CPipe::getDefinition (struct SDef *def*, s_read_style *sty*) [virtual]

Get the stream definition structure from xml parser and open file.

Parameters:

def Filled stream definition structure (input)

sty Style of reading (by length, or by start and end character)

Returns:

s_ret value about opening the port and about the

Implements **IStreamDaq** (p. 109).

10.6.2.2 s_ret CPipe::retStream (unsigned char ** *buffer*, unsigned long * *length*) [virtual]

Blocking method to wait the data to be passed to formatter (maybe after a decode and a "divide in message" processes respectively to interpret the escape charactera and to divide packets to the correct length).

Parameters:

buffer Pointer to a pointer of a vector of byte to be formatted (output)

length Pointer to the packet lenght variable (output)

Returns:

This method return one of the possible value from s_ret

See also:

s_ret (p. 196)

Implements **IStreamDaq** (p. 109).

10.6.2.3 s_ret CPipe::Pipe_init (void) [private]

Connection to a certain pipe like a client.

Returns:

s_ret value about connecting to certain pipe

See also:

s_ret (p. 196)

10.6.2.4 void CPipe::Pipe_close (void) [private]

Closing the connection to the pipe.

10.6.2.5 s_ret CPipe::Pipe_read (unsigned char * *data*, unsigned int *data_size*) [private]

Receive from pipe.

Parameters:

data Buffer to receive data

data_size Size of data to receive

Returns:

s_ret value about receiving from

See also:

s_ret (p. 196)

10.6.2.6 s_ret CPipe::Pipe_write (unsigned char * *data*, unsigned int *data_size*) [private]

Send by the pipe.

Parameters:

data Data buffer to send

data_size Size of data to send

Returns:

s_ret value about sending from

See also:

s_ret (p. 196)

10.6.2.7 s_ret CPipe::Pipe_handshaking (handshake_style *style*) [private]

This function execute the handshaking to initialize the stream.

Parameters:

style Handshacking style

Returns:

s_ret value about sending from

See also:

s_ret (p. 196)

10.6.3 Member Data Documentation

10.6.3.1 int CPipe::pfd[2] [private]

Pair of file descriptors, pointing to a pipe inode, and places them in the array pointed to by filedes. filedes[0] is for reading, filedes[1] is for writing.

10.6.3.2 pid_t CPipe::cpid [private]

Pipe handle

10.6.3.3 char* CPipe::pi_Name [private]

Pipe name

10.6.3.4 unsigned long CPipe::pi_Size [private]

Packet total size

10.6.3.5 unsigned long CPipe::pi_Size_r [private]

Received packet total size

10.6.3.6 s_read_style CPipe::pi_Style [private]

Reading style

10.6.3.7 handshake_style CPipe::pi_HStyle [private]

Handsaking style

10.6.3.8 unsigned char CPipe::pi_Start [private]

Start character

10.6.3.9 unsigned char CPipe::pi_End [private]

End character

10.6.3.10 std::vector<struct SHandshaking>* CPipe::pi_Handshk [private]

Handshaking packets

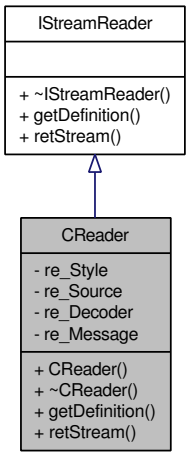
The documentation for this class was generated from the following files:

- **CPipe.h**
- **CPipe.cpp**

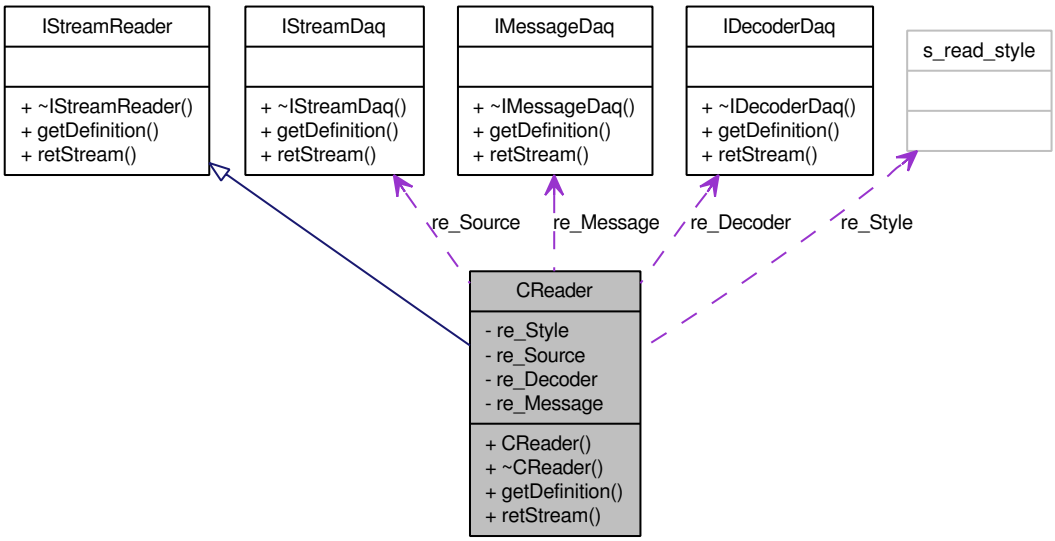
10.7 CReader Class Reference

Implemenation for **IStreamReader** (p. 110) interface to read, decode and divide packet to be passed to formatter. This class uses instantiation of the classes **IStreamDaq** (p. 108), **IDecoderDaq** (p. 102), **IMessageDaq** (p. 106).

Inheritance diagram for CReader:



Collaboration diagram for CReader:



Public Member Functions

- virtual **s_ret getDefinition** (struct **SDef** def)
- virtual **s_ret retStream** (std::vector< struct **SMessage** > *buffer)

Private Attributes

- **s_read_style re_Style**
- **IStreamDaq * re_Source**
- **IDecoderDaq * re_Decoder**
- **IMessageDaq * re_Message**

10.7.1 Constructor & Destructor Documentation

10.7.1.1 CReader::CReader ()

CReader (p. 61) constructor.

10.7.1.2 CReader::~CReader ()

CReader (p. 61) destructor.

10.7.2 Member Function Documentation

10.7.2.1 s_ret CReader::getDefinition (struct SDef *def*) [virtual]

Initialize all the classes linked with reader with the stream definition.

Returns:

This method return one of the possible value from s_ret

See also:

s_ret (p. 196)

Todo

usb

Todo

stdin

Implements **IStreamReader** (p. 110).

10.7.2.2 s_ret CReader::retStream (std::vector< struct SMessage > * *buffer*) [virtual]

This method divides the packet recived into message to be formatted (the lenght of each message is the pkt_lenght defined in the xml file).

Parameters:

buffer Pointer to a pointer of a vector of the messages to be formatted (output)

Returns:

This method return one of the possible value from s_ret

See also:

s_ret (p. 196)

Implements **IStreamReader** (p. 110).

10.7.3 Member Data Documentation

10.7.3.1 s_read_style CReader::re_Style [private]

Style of reading.

10.7.3.2 IStreamDaq* CReader::re_Source [private]

Source from wich data is read

10.7.3.3 IDecoderDaq* CReader::re_Decoder [private]

Type of decoder choosen

10.7.3.4 IMessageDaq* CReader::re_Message [private]

Class to transform pkts in msgs

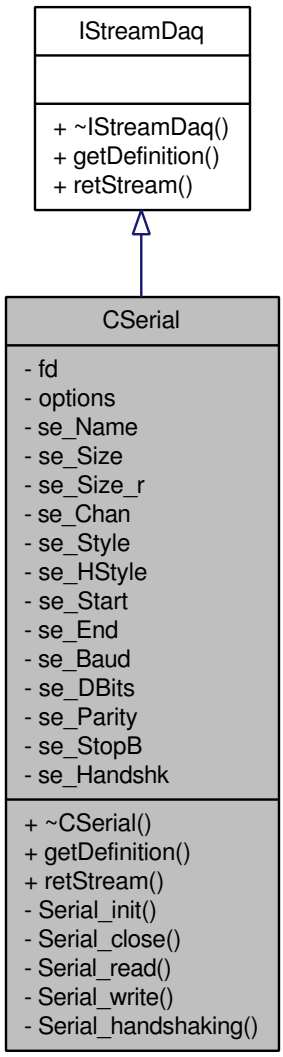
The documentation for this class was generated from the following files:

- **CReader.h**
- **CReader.cpp**

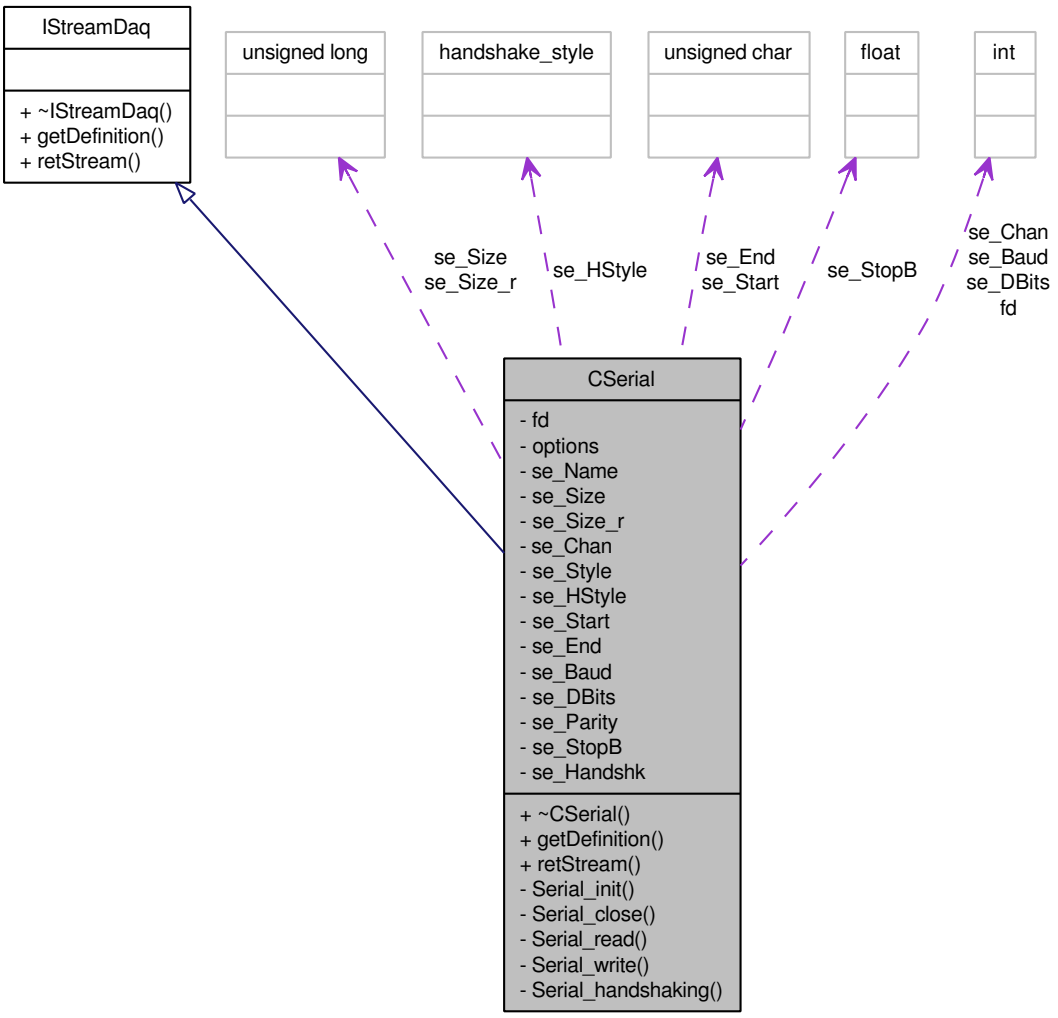
10.8 CSerial Class Reference

Implementation of IStream interface to read from serial (RS232). This class reads a file and creates packet shaped like the **SDef** (p. 116) structure directives.

Inheritance diagram for CSerial:



Collaboration diagram for CSerial:



Public Member Functions

- virtual **s_ret getDefinition** (struct **SDef** def, **s_read_style** sty)
- virtual **s_ret retStream** (unsigned char **buffer, unsigned long *length)

Private Member Functions

- **s_ret Serial_init** (void)
- **s_ret Serial_read** (unsigned char *data, unsigned int data_size)
- **s_ret Serial_write** (unsigned char *data, unsigned int data_size)
- **s_ret Serial_handshaking** (**handshake_style** style)

Private Attributes

- int **fd**

- struct termios **options**
- char * **se_Name**
- unsigned long **se_Size**
- unsigned long **se_Size_r**
- unsigned int **se_Chan**
- **s_read_style se_Style**
- **handshake_style se_HStyle**
- unsigned char **se_Start**
- unsigned char **se_End**
- unsigned int **se_Baud**
- unsigned int **se_DBits**
- std::string **se_Parity**
- float **se_StopB**
- std::vector< struct **SHandshaking** > * **se_Handshk**

10.8.1 Constructor & Destructor Documentation

10.8.1.1 CSerial::~CSerial ()

CSerial (p. 64) destructor.

10.8.2 Member Function Documentation

10.8.2.1 s_ret CSerial::getDefinition (struct SDef *def*, s_read_style *sty*) [virtual]

Get the stream definition structure from xml parser and open file.

Parameters:

def Filled stream definition structure (input)

sty Style of reading (by length, or by start and end character)

Returns:

s_ret value about opening the port and about the

Implements **IStreamDaq** (p. 109).

10.8.2.2 s_ret CSerial::retStream (unsigned char ** *buffer*, unsigned long * *length*) [virtual]

Blocking method to wait the data to be passed to formatter (maybe after a decode and a "divide in message" processes respectively to interpret the escape charactera and to divide packets to the correct length).

Parameters:

buffer Pointer to a pointer of a vector of byte to be formatted (output)

length Pointer to the packet lenght variable (output)

Returns:

This method return one of the possible value from s_ret

See also:

s_ret (p. 196)

Implements **IStreamDaq** (p. 109).

10.8.2.3 s_ret CSerial::Serial_init (void) [private]

Initialization of serial port.

Returns:

s_ret value about opening the serial port

See also:

s_ret (p. 196)

10.8.2.4 void CSerial::Serial_close (void) [private]

Closing of serial port.

10.8.2.5 s_ret CSerial::Serial_read (unsigned char * *data*, unsigned int *data_size*) [private]

Receive from serial port.

Parameters:

data Buffer to receive data

data_size Size of data to receive

Returns:

s_ret value about receiving from

See also:

s_ret (p. 196)

10.8.2.6 s_ret CSerial::Serial_write (unsigned char * *data*, unsigned int *data_size*) [private]

Send by the serial port.

Parameters:

data Data buffer to send

data_size Size of data to send

Returns:

s_ret value about sending from

See also:

s_ret (p. 196)

10.8.2.7 s_ret CSerial::Serial_handshaking (handshake_style *style*) [private]

This function execute the handshaking to initialize the stream.

Parameters:

style Handshacking style

Returns:

s_ret value about sending from

See also:

s_ret (p. 196)

10.8.3 Member Data Documentation

10.8.3.1 int CSerial::fd [private]

Handle to serial port

10.8.3.2 struct termios CSerial::options [read, private]

Posix structure for serial communication

10.8.3.3 char* CSerial::se_Name [private]

Serial name

10.8.3.4 unsigned long CSerial::se_Size [private]

Packet total size

10.8.3.5 unsigned long CSerial::se_Size_r [private]

Received packet total size

10.8.3.6 unsigned int CSerial::se_Chan [private]

Logical channel

10.8.3.7 s_read_style CSerial::se_Style [private]

Reading style

10.8.3.8 handshake_style CSerial::se_HStyle [private]

Handsaking style

10.8.3.9 unsigned char CSerial::se_Start [private]

Start character

10.8.3.10 unsigned char CSerial::se_End [private]

End character

10.8.3.11 unsigned int CSerial::se_Baud [private]

Baud rate

10.8.3.12 unsigned int CSerial::se_DBits [private]

Number of data bits

10.8.3.13 std::string CSerial::se_Parity [private]

Parity bit

10.8.3.14 float CSerial::se_StopB [private]

Number of stop bits

10.8.3.15 std::vector<struct SHandshaking>* CSerial::se_Handshk [private]

Handshaking packets to start communication

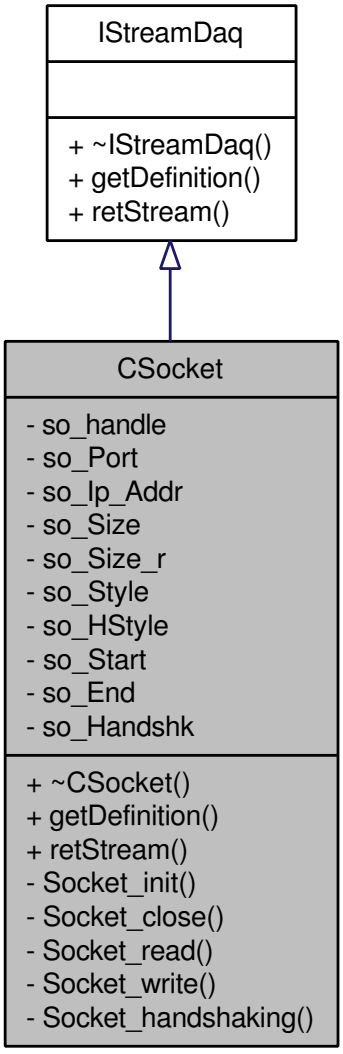
The documentation for this class was generated from the following files:

- **CSerial.h**
- **CSerial.cpp**

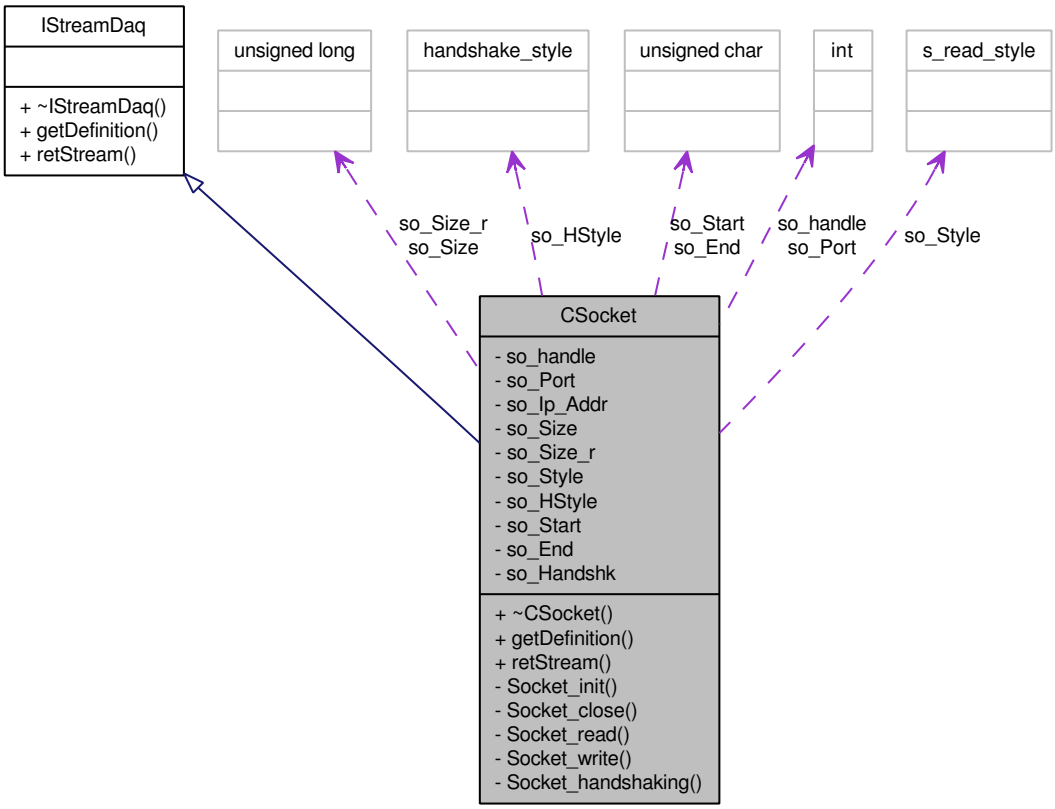
10.9 CSocket Class Reference

Implementation of IStream interface to read a stream from a socket. This class bytes form socket and creates packet shaped like the **SDef** (p. 116) structure directives.

Inheritance diagram for CSocket:



Collaboration diagram for CSocket:



Public Member Functions

- virtual **s_ret getDefinition** (struct **SDef** def, **s_read_style** sty)
- virtual **s_ret retStream** (unsigned char **buffer, unsigned long *length)

Private Member Functions

- **s_ret Socket_init** (void)
- **s_ret Socket_read** (unsigned char *data, unsigned int data_size)
- **s_ret Socket_write** (unsigned char *data, unsigned int data_size)
- **s_ret Socket_handshaking** (**handshake_style** style)

Private Attributes

- int **so_handle**
- unsigned int **so_Port**
- char * **so_Ip_Addr**
- unsigned long **so_Size**
- unsigned long **so_Size_r**
- **s_read_style so_Style**
- **handshake_style so_HStyle**

- unsigned char **so_Start**
- unsigned char **so_End**
- std::vector< struct **SHandshaking** > * **so_Handshk**

10.9.1 Constructor & Destructor Documentation

10.9.1.1 CSocket::~CSocket ()

CPipe (p. 55) destructor.

10.9.2 Member Function Documentation

10.9.2.1 s_ret CSocket::getDefinition (struct SDef *def*, s_read_style *sty*) [virtual]

Get the stream definition structure from xml parser and open file.

Parameters:

def Filled stream definition structure (input)

sty Style of reading (by length, or by start and end character)

Returns:

s_ret value about opening the port and about the

Implements **IStreamDaq** (p. 109).

10.9.2.2 s_ret CSocket::retStream (unsigned char ** *buffer*, unsigned long * *length*) [virtual]

Blocking method to wait the data to be passed to formatter (maybe after a decode and a "divide in message" processes respectively to interpret the escape charactera and to divide packets to the correct length).

Parameters:

buffer Pointer to a pointer of a vector of byte to be formatted (output)

length Pointer to the packet lenght variable (output)

Returns:

This method return one of the possible value from s_ret

See also:

s_ret (p. 196)

Implements **IStreamDaq** (p. 109).

10.9.2.3 s_ret CSocket::Socket_init (void) [private]

Connection to a certain port socket like a client.

Returns:

s_ret value about connecting to certain port socket

See also:

s_ret (p. 196)

10.9.2.4 void CSocket::Socket_close (void) [private]

Closing the socket connection.

10.9.2.5 s_ret CSocket::Socket_read (unsigned char * *data*, unsigned int *data_size*) [private]

Receive from socket.

Parameters:

data Buffer to receive data

data_size Size of data to receive

Returns:

s_ret value about receiving from

See also:

s_ret (p. 196)

10.9.2.6 s_ret CSocket::Socket_write (unsigned char * *data*, unsigned int *data_size*) [private]

Send by socket.

Parameters:

data Data buffer to send

data_size Size of data to send

Returns:

s_ret value about sending from

See also:

s_ret (p. 196)

10.9.2.7 s_ret CSocket::Socket_handshaking (handshake_style *style*) [private]

This function execute the handshaking to initialize the stream.

Parameters:

style Handshacking style

Returns:

s_ret value about sending from

See also:

s_ret (p. 196)

10.9.3 Member Data Documentation

10.9.3.1 int CSocket::so_handle [private]

Socket handle

10.9.3.2 unsigned int CSocket::so_Port [private]

Port number

10.9.3.3 char* CSocket::so_Ip_Addr [private]

Server IP address

10.9.3.4 unsigned long CSocket::so_Size [private]

Packet total size

10.9.3.5 unsigned long CSocket::so_Size_r [private]

Received packet total size

10.9.3.6 s_read_style CSocket::so_Style [private]

Reading style

10.9.3.7 handshake_style CSocket::so_HStyle [private]

Handsaking style

10.9.3.8 unsigned char CSocket::so_Start [private]

Start character

10.9.3.9 unsigned char CSocket::so_End [private]

End character

10.9.3.10 std::vector<struct SHandshaking>* CSocket::so_Handshk [private]

Handshaking packets

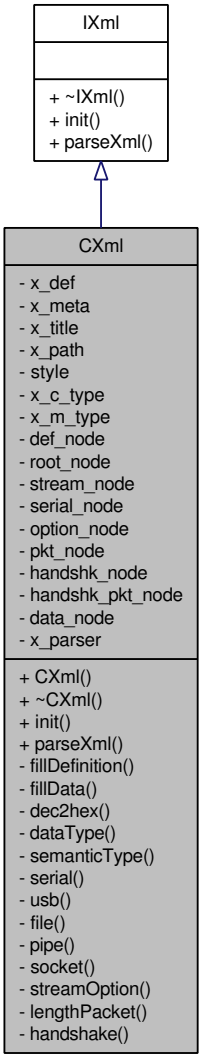
The documentation for this class was generated from the following files:

- **CSocket.h**
- **CSocket.cpp**

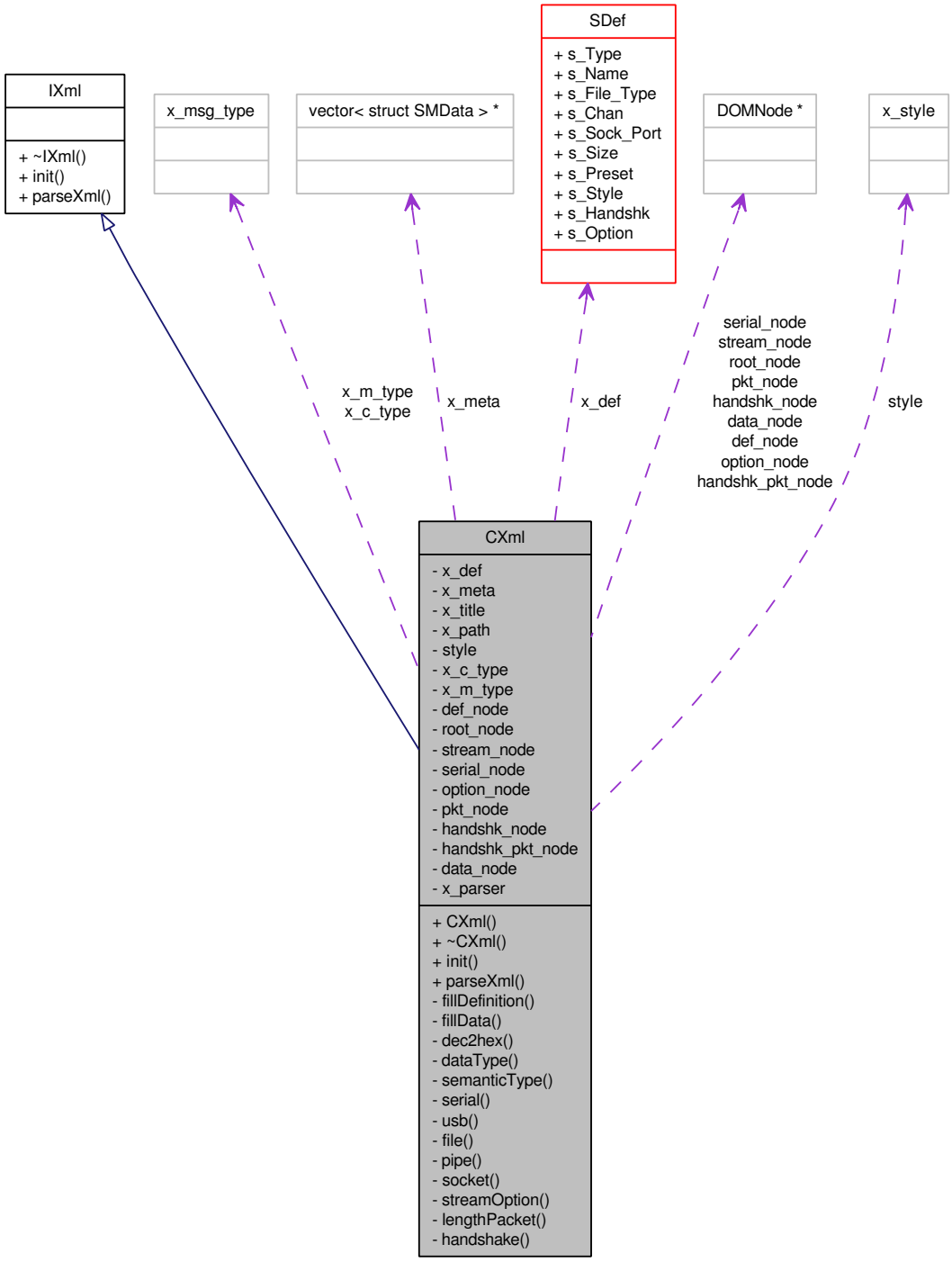
10.10 CXml Class Reference

DAQ specific xml parsing class: class for xml parsing using xml_schema input.xsd.

Inheritance diagram for CXml:



Collaboration diagram for CXml:



Public Member Functions

- **CXml** (char *path_xml, struct **SDef** *def, vector< struct **SMData** > *metadata, string *title)
- virtual **x_ret init** ()

- virtual **x_ret parseXml** ()

Private Member Functions

- **x_ret fillDefinition** (void)
- **x_ret fillData** (void)
- unsigned char **dec2hex** (char ls, char hs)
- **var_type dataType** (char *ret, unsigned long size)
- **data_semantic semanticType** (char *ret)
- **x_ret serial** (void)
- **x_ret usb** (void)
- **x_ret file** (void)
- **x_ret pipe** (void)
- **x_ret socket** (void)
- **x_ret streamOption** (void)
- **x_ret lengthPacket** (void)
- **x_ret handshake** (void)

Private Attributes

- struct **SDef** * **x_def**
- vector< struct **SMData** > * **x_meta**
- string * **x_title**
- const char * **x_path**
- **x_style style**
- **x_msg_type x_c_type**
- **x_msg_type x_m_type**
- DOMNode * **def_node**
- DOMNode * **root_node**
- DOMNode * **stream_node**
- DOMNode * **serial_node**
- DOMNode * **option_node**
- DOMNode * **pkt_node**
- DOMNode * **handshk_node**
- DOMNode * **handshk_pkt_node**
- DOMNode * **data_node**
- **CXMLParser** * **x_parser**

10.10.1 Constructor & Destructor Documentation

10.10.1.1 CXml::CXml (char * *path_xml*, struct SDef * *def*, vector< struct SMData > * *metadata*, string * *title*)

Constructor.

Parameters:

path_xml Xml file path (input)

def Pointer to stream definition structure (output)

metadata Pointer to metadata vector (output)

10.10.1.2 CXml::~CXml ()

Destructor.

10.10.2 Member Function Documentation

10.10.2.1 x_ret CXml::init (void) [virtual]

Initialize the parser.

Returns:

This method return one of the possible value from enum

Implements **IXml** (p. 113).

10.10.2.2 x_ret CXml::parseXml () [virtual]

This method make the parsing operation and fill def and metadata (validating the parsed value).

Parameters:

title This is a pointer to a variable contains the title of the output vector (output)

Returns:

This method return one of the possible value from enum

Implements **IXml** (p. 113).

10.10.2.3 x_ret CXml::fillDefinition (void) [private]

Check if the stream definition is correct and fill x_def and title.

Returns:

This method return one of the possible value from enum

See also:

x_ret (p. 199)

10.10.2.4 x_ret CXml::fillData (void) [private]

Check if the data definition is correct and fill x_meta.

Returns:

This method return one of the possible value from enum

See also:

x_ret (p. 199)

10.10.2.5 unsigned char CXml::dec2hex (char *ls*, char *hs*) [private]

Convert decimal value (write like a sting of char) into hexadecimal value.

Parameters:

ls lowest signifivative value
hs highest signifivative value

Returns:

return the hexadecimal value

10.10.2.6 var_type CXml::dataType (char * *ret*, unsigned long *size*) [private]

Convert a string (char*) in type var_type.

Parameters:

ret string that contains the value read from the xml-tree
size total size in byte of the field

Returns:

return one of the type of var_type

10.10.2.7 data_semantic CXml::semanticType (char * *ret*) [private]

Convert a string (char*) in type data_semantic.

Parameters:

ret string that contains the value read from the xml-tree

Returns:

return one of the semantic type of data_semantic

10.10.2.8 x_ret CXml::serial (void) [private]

Fill the preset field of x_def for serial.

Returns:

return one of the x_ret value

10.10.2.9 x_ret CXml::usb (void) [private]

Fill the preset field of x_def for usb.

Returns:

return one of the x_ret value

10.10.2.10 x_ret CXml::file (void) [private]

Fill the preset field of x_def for file.

Returns:

return one of the x_ret value

10.10.2.11 x_ret CXml::pipe (void) [private]

Fill the preset field of x_def for pipe.

Returns:

return one of the x_ret value

10.10.2.12 x_ret CXml::socket (void) [private]

Fill the preset field of x_def for socket.

Returns:

return one of the x_ret value

10.10.2.13 x_ret CXml::streamOption (void) [private]

Fill the structure x_def->s_Option.

See also:

SOption (p. 124)

Returns:

return one of the x_ret value

10.10.2.14 x_ret CXml::lengthPacket (void) [private]

Read from xml tree the size of the packet.

Returns:

return one of the x_ret value

10.10.2.15 x_ret CXml::handshake (void) [private]

Fill the structure x_def->s_Handshk.

See also:

SHandshaking (p. 119)

Returns:

return one of the x_ret value

10.10.3 Member Data Documentation

10.10.3.1 struct SDef* CXml::x_def [read, private]

SDef (p. 116) variable to fill with the data parsed from xml file

10.10.3.2 vector<struct SMData>* CXml::x_meta [private]

Metadata vector

10.10.3.3 string* CXml::x_title [private]

Title

10.10.3.4 const char* CXml::x_path [private]

Path of xml file

10.10.3.5 x_style CXml::style [private]

Choosen xml file style

See also:

x_style (p. 200)

10.10.3.6 x_msg_type CXml::x_c_type [private]

Character type for special character

See also:

x_msg_type (p. 200)

10.10.3.7 x_msg_type CXml::x_m_type [private]

Character type for handshake msg

See also:

x_msg_type (p. 200)

10.10.3.8 DOMNode* CXml::def_node [private]

DOMDocument pointer

10.10.3.9 DOMNode* CXml::root_node [private]

DOMNode pointer for root node

10.10.3.10 DOMNode* CXml::stream_node [private]

DOMNode pointer for stream node

10.10.3.11 DOMNode* CXml::serial_node [private]

DOMNode pointer for serial serial presets node

10.10.3.12 DOMNode* CXml::option_node [private]

DOMNode pointer for serial stream option node

10.10.3.13 DOMNode* CXml::pkt_node [private]

DOMNode pointer for packet description

10.10.3.14 DOMNode* CXml::handshk_node [private]

DOMNode pointer for handshaking node

10.10.3.15 DOMNode* CXml::handshk_pkt_node [private]

DOMNode pointer for handshaking packet node

10.10.3.16 DOMNode* CXml::data_node [private]

DOMNode pointer for data node

10.10.3.17 CXMLParser* CXml::x_parser [private]

Pointer to Xml parser

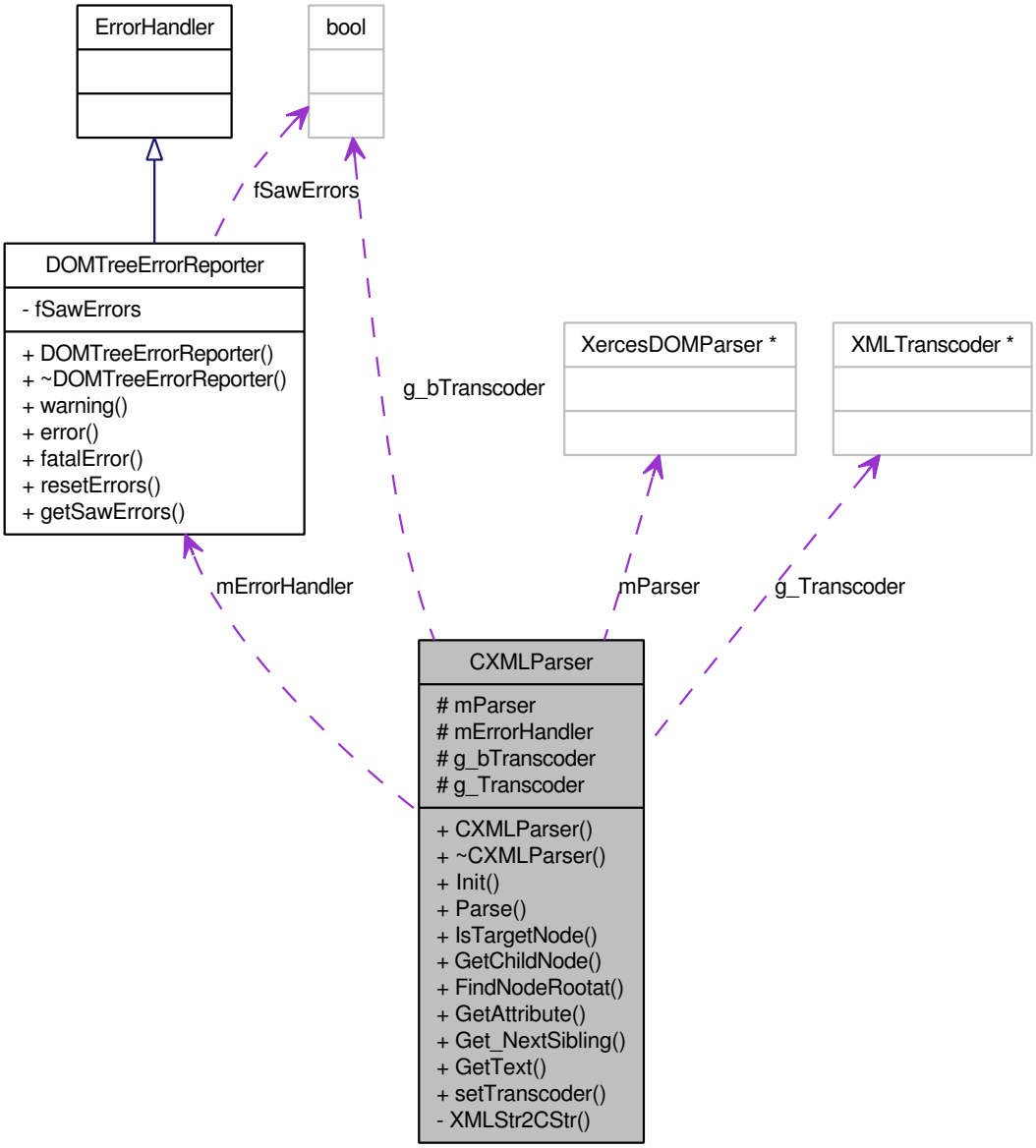
The documentation for this class was generated from the following files:

- **CXml.h**
- **CXml.cpp**

10.11 CXMLParser Class Reference

Class to implement methods to init, parse, process an xml file. Genercic class.

Collaboration diagram for CXMLParser:



Public Member Functions

- **xml_parser_ret_value Init** (void)
- **xml_parser_ret_value Parse** (char *fn, DOMNode *&doc)
- bool **IsTargetNode** (DOMNode *node, char *targetname)
- bool **GetChildNode** (DOMNode *curRoot, char *targetname, DOMNode *&targetnode)
- bool **FindNodeRootat** (DOMNode *curRoot, char *targetname, DOMNode *&targetnode)

- bool **GetAttribute** (DOMNode *node, char *attrname, char *attrvalue)
- DOMNode * **Get_NextSibling** (DOMNode *node)
- bool **GetText** (DOMNode *node, char *text)
- void **setTranscoder** (char *codepage)

Protected Attributes

- XercesDOMParser * **mParser**
- **DOMTreeErrorReporter** * **mErrorHandler**
- bool **g_bTranscoder**
- XMLTranscoder * **g_Transcoder**

Private Member Functions

- char * **XMLStr2CStr** (const XMLCh *s)

10.11.1 Constructor & Destructor Documentation

10.11.1.1 CXMLParser::CXMLParser ()

CXMLParser (p. 85) contructor.

10.11.1.2 CXMLParser::~CXMLParser ()

CXMLParser (p. 85) virtual distructor.

10.11.2 Member Function Documentation

10.11.2.1 char * CXMLParser::XMLStr2CStr (const XMLCh * *s*) [private]

Method to decode string from XMLCh* to char*.

Parameters:

s Xml string (input)

Returns:

Decoded string

10.11.2.2 xml_parser_ret_value CXMLParser::Init (void)

Initialize the parser.

Parameters:

bValidate true if is requested the xml file validation (input)

Returns:

Returns an xml_parser_ret_value as a result of initialization operation

See also:

xml_parser_ret_value (p. 198)

10.11.2.3 xml_parser_ret_value CXMLParser::Parse (char * *fn*, DOMNode *& *doc*)

Parse xml file.

Parameters:

fn filename (input)

doc DOC node pointer: this node will be the root of the parsing tree (output)

Returns:

Returns an xml_parser_ret_value as a result of parsing operation

See also:

xml_parser_ret_value (p. 198)

10.11.2.4 bool CXMLParser::IsTargetNode (DOMNode * *node*, char * *targetname*)

Check if the actual node corresponds with the required one.

Parameters:

node Actual DOM node pointer (input)

targetname Name of the target DOM node (input)

Returns:

Returns true if the actual node is the target node, false otherwise

10.11.2.5 bool CXMLParser::GetChildNode (DOMNode * *curRoot*, char * *targetname*, DOMNode *& *targetnode*)

Get the child node of curRoot.

Parameters:

curRoot Actual DOM node (root) pointer (input)

targetname Name of the target (child) DOM node (input)

targetnode Child DOM node pointer (output)

Returns:

Returns true if the child node is the target node, false otherwise

10.11.2.6 bool CXMLParser::FindNodeRootat (DOMNode * *curRoot*, char * *targetname*, DOMNode *& *targetnode*)

Search a target node inside the tree.

Parameters:

curRoot Actual DOM node (root) pointer(input)

targetname Name of the target (child) DOM node (input)

targetnode Child DOM node pointer (output)

Returns:

Returns true if target node is found, false otherwise

10.11.2.7 bool CXMLParser::GetAttribute (DOMNode * *node*, char * *attrname*, char * *attrvalue*)

Get the value of the attribute called "attrname" of the current node (DOMNode* node).

Parameters:

node Target node (input)

attrname Attribute name (input)

attrvalue Attribute value (output)

Returns:

Returns true the attribute called attrname is found, false otherwise

10.11.2.8 DOMNode * CXMLParser::Get_NextSibling (DOMNode * *node*)

Find the sibling node with ELEMENT_NODE (=1) type.

Parameters:

node DOMNode pointer (input)

Returns:

Returns a pointer of DOMNode class of the next sibling node, if it exists; NULL if it doesn't exist. This function is very usefull because a lot of parsers treat empty white-spaces or new lines as text nodes; in this way is possible to find only the ELEMENT_NODE

10.11.2.9 bool CXMLParser::GetText (DOMNode * *node*, char * *text*)

Get the text of the child text node of actual node (DOMNode* node).

Parameters:

node Actual node (input)

text Text (output)

Returns:

Returns true if the text node is found, false otherwise

10.11.2.10 void CXMLParser::setTranscoder (char * *codepage*)

Change string encoder.

Parameters:

codepage Encoding type (input)

10.11.3 Member Data Documentation

10.11.3.1 XercesDOMParser* CXMLParser::mParser [protected]

XercesDOMParser class object

10.11.3.2 DOMTreeErrorReporter* CXMLParser::mErrorHandler [protected]

DOMTreeErrorReporter (p. 98) class object

10.11.3.3 bool CXMLParser::g_bTranscoder [protected]

brief false if standard encoding (UTF-16) is used, true otherwise

10.11.3.4 XMLTranscoder* CXMLParser::g_Transcoder [protected]

brief Transcoder pointer

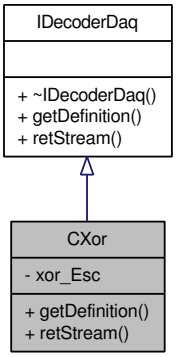
The documentation for this class was generated from the following files:

- **CXmlParser.h**
- **CXmlParser.cpp**

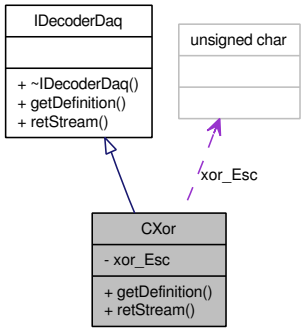
10.12 CXor Class Reference

Implementation of **IDecoderDaq** (p. 102) interface to decode the escape character with xor function.

Inheritance diagram for CXor:



Collaboration diagram for CXor:



Public Member Functions

- virtual **s_ret getDefinition** (struct **SDef** def)
- virtual **s_ret retStream** (unsigned char *buffer_i, unsigned long length_i, unsigned char **buffer_o, unsigned long *length_o)

Private Attributes

- unsigned char **xor_Esc**

10.12.1 Member Function Documentation

10.12.1.1 s_ret CXor::getDefinition (struct SDef *def*) [virtual]

Initialize the stream decoder.

Parameters:

def Pointer to a stream definition structure (input)

Returns:

This method return one of the possible value from s_ret

See also:

s_ret (p. 196)

Implements **IDecoderDaq** (p. 102).

10.12.1.2 s_ret CXor::retStream (unsigned char * *buffer_i*, unsigned long *length_i*, unsigned char ** *buffer_o*, unsigned long * *length_o*) [virtual]

Method that return the decoded packet.

Parameters:

buffer_i Pointer of the buffet to decode (input)

length_i Length of the buffer to decode (input)

buffer_o Pointer to a pointer of a vector of byte to be formatted (output)

length_o Pointer to the packet lenght variable (output)

Returns:

This method return one of the possible value from s_ret

See also:

s_ret (p. 196)

Implements **IDecoderDaq** (p. 103).

10.12.2 Member Data Documentation

10.12.2.1 unsigned char CXor::xor_Esc [private]

Escape character

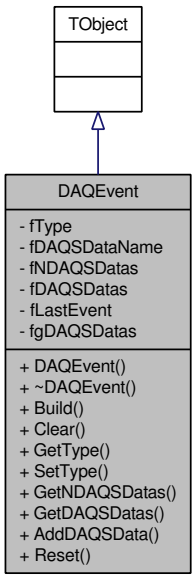
The documentation for this class was generated from the following files:

- **CXor.h**
- **CXor.cpp**

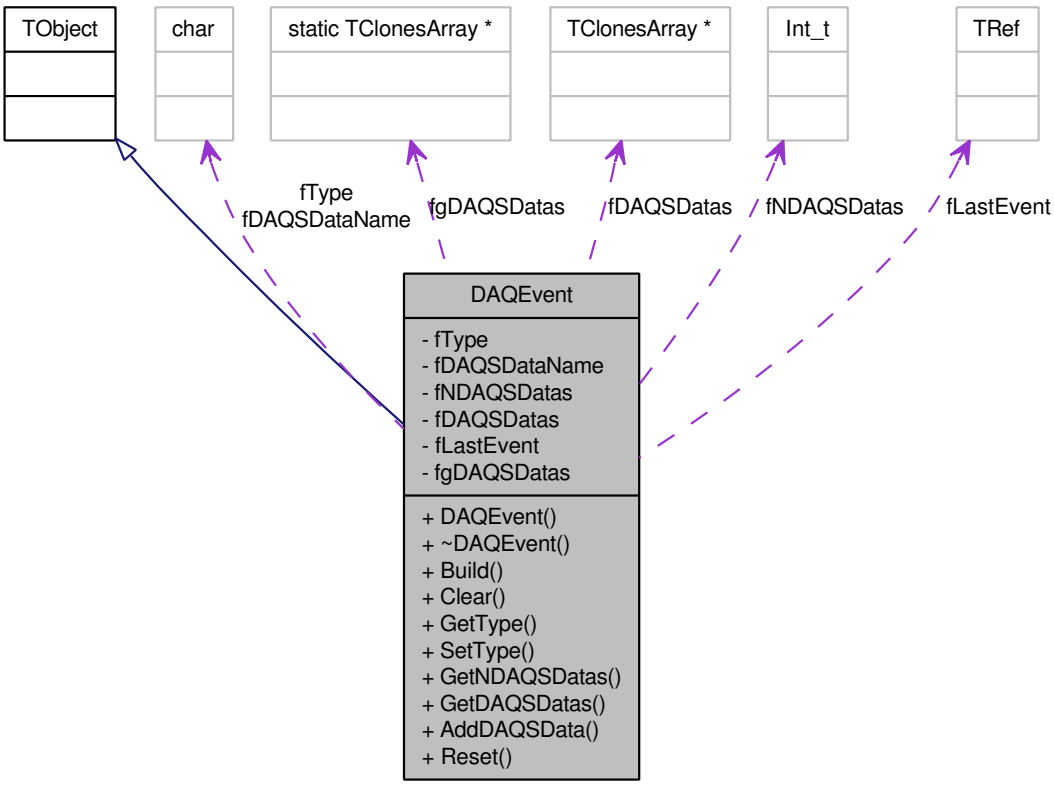
10.13 DAQEvent Class Reference

Class containing members and methods for a complete event acquired through the link.

Inheritance diagram for DAQEvent:



Collaboration diagram for DAQEvent:



Public Member Functions

- void **Build** (Int_t ev, std::vector< std::vector< **SData** > > *data)

Private Attributes

- char **fType** [20]
- char * **fDAQSDataName**
- Int_t **fNDAQSDatas**
- TClonesArray * **fDAQSDatas**
- TRef **fLastEvent**

Static Private Attributes

- static TClonesArray * **fgDAQSDatas**

10.13.1 Constructor & Destructor Documentation

10.13.1.1 DAQEvent::DAQEvent ()

10.13.1.2 DAQEvent::~DAQEvent () [virtual]

10.13.2 Member Function Documentation

10.13.2.1 void DAQEvent::Build (Int_t *ev*, std::vector< std::vector< SData > > * *data*)

build a new event

Parameters:

ev Event number

nsdata Number of **SData** (p. 114)

a Pointer to **SData** (p. 114)

10.13.2.2 void DAQEvent::Clear (Option_t * *option* = "")

10.13.2.3 void DAQEvent::Reset (Option_t * *option* = "") [static]

10.13.2.4 char* DAQEvent::GetType ()

10.13.2.5 void DAQEvent::SetType (char * *type*)

10.13.2.6 Int_t DAQEvent::GetNDAQSDatas () const

10.13.2.7 TClonesArray* DAQEvent::GetDAQSDatas () const

10.13.2.8 DAQSData * DAQEvent::AddDAQSData (std::vector< SData > *sdata_*)

10.13.3 Member Data Documentation

10.13.3.1 char DAQEvent::fType[20] [private]

String containing the type of the event

10.13.3.2 char* DAQEvent::fDAQSDataName [private]

Run+event number in character format

10.13.3.3 Int_t DAQEvent::fNDAQSDatas [private]

Number of **SData** (p. 114) objects in the event

10.13.3.4 TClonesArray* DAQEvent::fDAQSDatas [private]

Array with all the events stored

10.13.3.5 TRef DAQEvent::fLastEvent [private]

reference pointer to last **SData** (p. 114)

10.13.3.6 TClonesArray* DAQEvent::fgDAQSDatas [static, private]

static counterpart of fDAQSDatas

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

- **DAQEvent.h**
- **DAQEvent.cxx**

10.14 DAQPayload Struct Reference

Structure that describe the packet payload according with xml file.

10.14.1 Detailed Description

Warning:

This structure has to be modified everytime the xml file payload description is changed

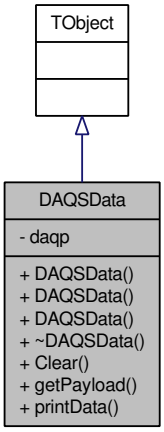
The documentation for this struct was generated from the following file:

- **DAQPayload.h**

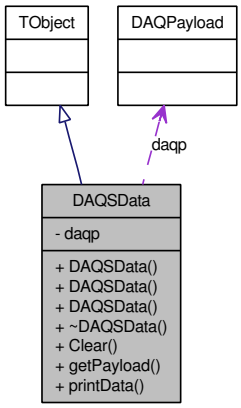
10.15 DAQSData Class Reference

Class containing members and methods for the data acquired through the link.

Inheritance diagram for DAQSData:



Collaboration diagram for DAQSData:



Private Attributes

- **DAQPayload daqp**

10.15.1 Detailed Description

Warning:

This class has to be modified everytime the xml file payload description is changed

10.15.2 Constructor & Destructor Documentation

10.15.2.1 DAQSData::DAQSData ()

10.15.2.2 DAQSData::DAQSData (const DAQSData & *orig*)

10.15.2.3 DAQSData::DAQSData (std::vector< SData > *sdata_*)

10.15.2.4 virtual DAQSData::~DAQSData () [virtual]

10.15.3 Member Function Documentation

10.15.3.1 void DAQSData::Clear (Option_t * *option* = "")

10.15.3.2 DAQPayload* DAQSData::getPayload (void)

returns the raw data

10.15.3.3 void DAQSData::printData () const

prints the data as a table

10.15.4 Member Data Documentation

10.15.4.1 DAQPayload DAQSData::daqp [private]

Member raw data

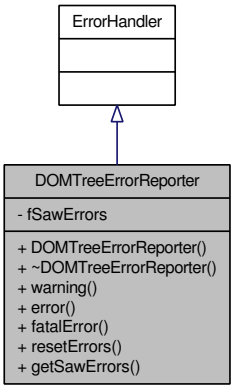
The documentation for this class was generated from the following files:

- **DAQSData.h**
- **DAQSData.cxx**

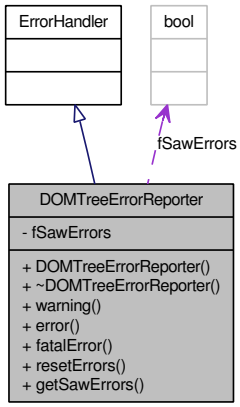
10.16 DOMTreeErrorReporter Class Reference

This class registers as an **ErrorHandler** (p. 101) with the DOM parser and reports errors to the application.

Inheritance diagram for DOMTreeErrorReporter:



Collaboration diagram for DOMTreeErrorReporter:



Public Member Functions

- void **warning** (const SAXParseException &toCatch)
- void **error** (const SAXParseException &toCatch)
- void **fatalError** (const SAXParseException &toCatch)
- bool **getSawErrors** () const

Private Attributes

- bool **fSawErrors**

10.16.1 Constructor & Destructor Documentation

10.16.1.1 DOMTreeErrorReporter::DOMTreeErrorReporter ()

Constructor.

10.16.1.2 DOMTreeErrorReporter::~DOMTreeErrorReporter ()

Destructor.

10.16.2 Member Function Documentation

10.16.2.1 void DOMTreeErrorReporter::warning (const SAXParseException & *toCatch*)

Warning handler.

Parameters:

toCatch Instance of SAX Parser error handler class

10.16.2.2 void DOMTreeErrorReporter::error (const SAXParseException & *toCatch*)

Error handler.

Parameters:

toCatch Instance of SAX Parser error handler class

10.16.2.3 void DOMTreeErrorReporter::fatalError (const SAXParseException & *toCatch*)

Fatal error handler.

Parameters:

toCatch Instance of SAX Parser error handler class

10.16.2.4 void DOMTreeErrorReporter::resetErrors ()

Reset error.

10.16.2.5 bool DOMTreeErrorReporter::getSawErrors () const

Getter methods.

Returns:

True if an error was found

10.16.3 Member Data Documentation

10.16.3.1 bool DOMTreeErrorReporter::fSawErrors [private]

This is set true if we get any errors, and is queryable via a getter method.

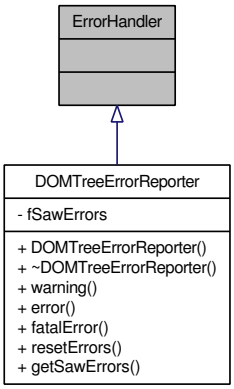
The documentation for this class was generated from the following files:

- **DOMTreeErrorReporter.hpp**
- **DOMTreeErrorReporter.cpp**

10.17 ErrorHandler Class Reference

Basic interface for SAX error handlers. From the Xerces package.

Inheritance diagram for ErrorHandler:



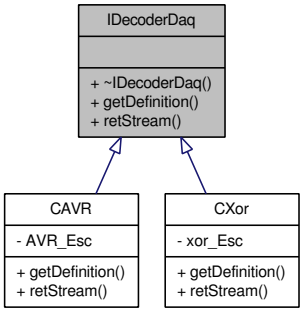
The documentation for this class was generated from the following file:

- **DOMTreeErrorReporter.hpp**

10.18 IDecoderDaq Class Reference

Interface for the decoder classes. Decoders need to decrypt a packet received or to resolve the escape charatecter inside the packets. This class is used by the **CReader** (p. 61) class.

Inheritance diagram for IDecoderDaq:



Public Member Functions

- virtual **s_ret getDefinition** (struct **SDef** def)=0
- virtual **s_ret retStream** (unsigned char *buffer_i, unsigned long length_i, unsigned char **buffer_o, unsigned long *length_o)=0

10.18.1 Constructor & Destructor Documentation

10.18.1.1 virtual IDecoderDaq::~IDecoderDaq () [virtual]

10.18.2 Member Function Documentation

10.18.2.1 virtual s_ret IDecoderDaq::getDefinition (struct SDef *def*) [pure virtual]

Initialize the stream decoder.

Parameters:

def Pointer to a stream definition structure (input)

Returns:

This method return one of the possible value from s_ret

See also:

s_ret (p. 196)

Implemented in **CAVR** (p. 36), and **CXor** (p. 90).

10.18.2.2 virtual s_ret IDecoderDaq::retStream (unsigned char * *buffer_i*, unsigned long *length_i*, unsigned char ** *buffer_o*, unsigned long * *length_o*) [pure virtual]

Method that return the decoded packet.

Parameters:

buffer_i Pointer of the buffet to decode (input)

length_i Length of the buffer to decode (input)

buffer_o Pointer to a pointer of a vector of byte to be formatted (output)

length_o Pointer to the packet lenght variable (output)

Returns:

This method return one of the possible value from s_ret

See also:

s_ret (p. 196)

Implemented in **CAVR** (p. 36), and **CXor** (p. 91).

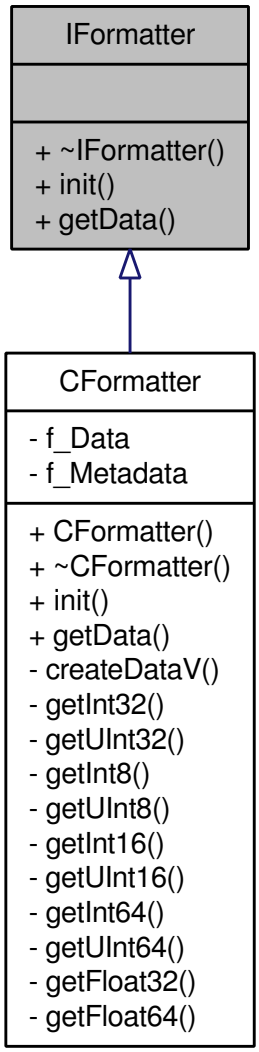
The documentation for this class was generated from the following file:

- **IDecoderDaq.h**

10.19 IFormatter Class Reference

Interface for data formatter.

Inheritance diagram for IFormatter:



Public Member Functions

- virtual **f_ret init** (vector< struct **SMData** > *metadata)=0
- virtual **f_ret getData** (unsigned char *pkt)=0

10.19.1 Constructor & Destructor Documentation

10.19.1.1 virtual IFormatter::~IFormatter () [virtual]

10.19.2 Member Function Documentation

10.19.2.1 virtual f_ret IFormatter::init (vector< struct SMData > * *metadata*) [pure virtual]

This method creates the header vector and the formatted data vector.

Parameters:

metadata Metadata vector

Returns:

One of the possible value of f_ret

See also:

f_ret (p. 175)

Implemented in **CFormatter** (p. 48).

10.19.2.2 virtual f_ret IFormatter::getData (unsigned char * *pkt*) [pure virtual]

Fill the vector of formatted data.

Parameters:

msg Pointer to a char vector of data to be formatted received from **IStreamReader** (p. 110) (input)

Returns:

One of the possible value of f_ret

See also:

f_ret (p. 175)

Implemented in **CFormatter** (p. 49).

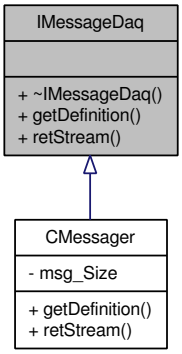
The documentation for this class was generated from the following file:

- **IFormatter.h**

10.20 IMessageDaq Class Reference

Interface for the messagger classes. Some application couldn't send single messages, but group of messages (for example tables): the aim of the classes those implement this inteface is to devide the data received in single messages. This class is used by the **CReader** (p. 61) class.

Inheritance diagram for IMessageDaq:



Public Member Functions

- virtual **s_ret getDefinition** (struct **SDef** def)=0
- virtual **s_ret retStream** (unsigned char *buffer_i, unsigned long length_i, std::vector< struct **SMessage** > *buffer_o)=0

10.20.1 Constructor & Destructor Documentation

10.20.1.1 virtual IMessageDaq::~IMessageDaq () [virtual]

10.20.2 Member Function Documentation

10.20.2.1 virtual s_ret IMessageDaq::getDefinition (struct SDef *def*) [pure virtual]

Initialize the messages maker.

Parameters:

def Pointer to a stream definition structure (input)

Returns:

This method return one of the possible value from s_ret

See also:

s_ret (p. 196)

Implemented in **CMessager** (p. 53).

10.20.2.2 virtual s_ret IMessageDaq::retStream (unsigned char * *buffer_i*, unsigned long *length_i*, std::vector< struct SMessage > * *buffer_o*) [pure virtual]

This method divides the packet recived into message to be formatted (the lenght of each message is the pkt_lenght defined in the xml file).

Parameters:

buffer_i Pointer of the buffet to decode (input)

length_i Length of the buffer to decode (input)

buffer_o Pointer to a vector of the messages to be formatted (output)

Returns:

This method return one of the possible value from s_ret

See also:

s_ret (p. 196)

Implemented in **CMessager** (p. 54).

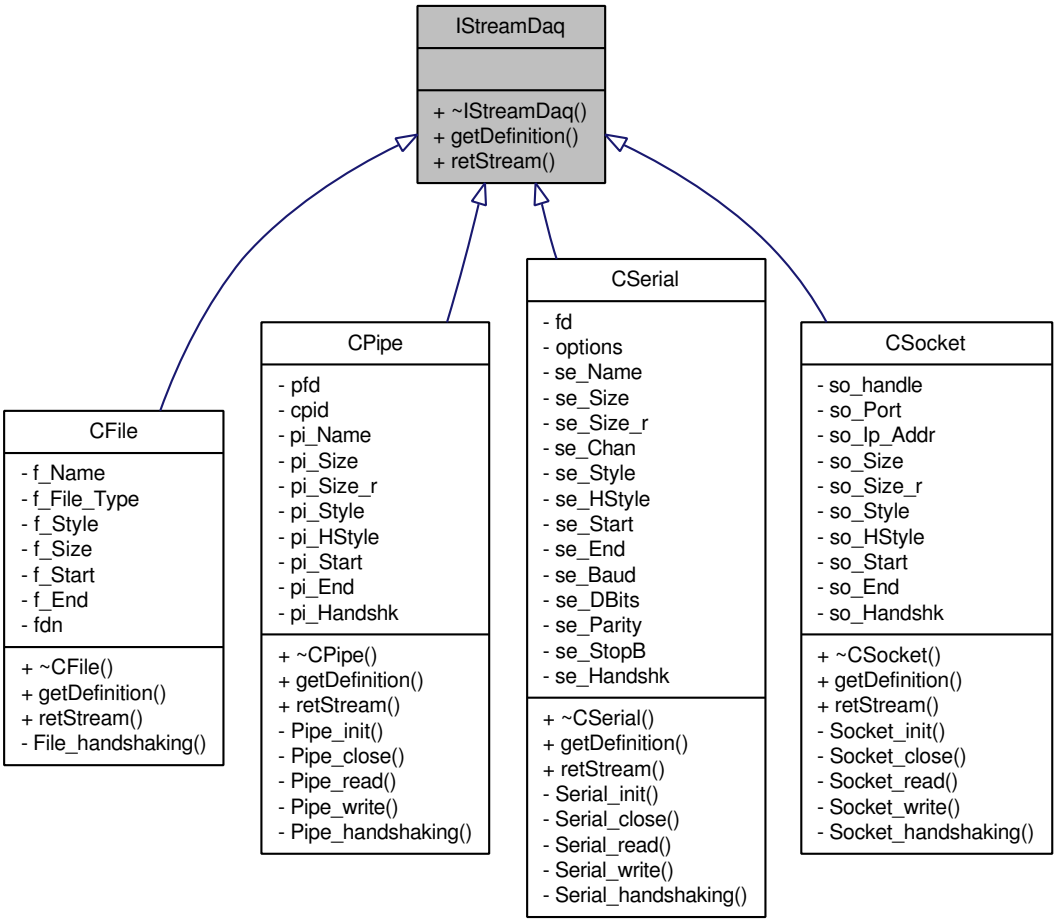
The documentation for this class was generated from the following file:

- **IMessageDaq.h**

10.21 IStreamDaq Class Reference

Interface for different type of stream. This interface is the model for the lowest layer classes that read data directly from differente devices. This class is used by the **CReader** (p. 61) class.

Inheritance diagram for IStreamDaq:



Public Member Functions

- virtual **s_ret getDefinition** (struct **SDef** def, **s_read_style** sty)=0
- virtual **s_ret retStream** (unsigned char **buffer, unsigned long *length)=0

10.21.1 Constructor & Destructor Documentation

10.21.1.1 virtual IStreamDaq::~IStreamDaq () [virtual]

10.21.2 Member Function Documentation

10.21.2.1 virtual s_ret IStreamDaq::getDefinition (struct SDef *def*, s_read_style *sty*) [pure virtual]

Initialize the stream reader.

Parameters:

def Pointer to a stream definition structure (input)

sty Style of reading (by length, or by start and end character)

Returns:

This method return one of the possible value from s_ret

See also:

s_ret (p. 196)

Implemented in **CFile** (p. 43), **CPipe** (p. 57), **CSerial** (p. 66), and **CSocket** (p. 72).

10.21.2.2 virtual s_ret IStreamDaq::retStream (unsigned char ** *buffer*, unsigned long * *length*) [pure virtual]

Blocking method to wait the data to be passed to formatter.

Parameters:

buffer Pointer to a pointer of a vector of byte to be formatted (output)

length Pointer to the packet lenght variable (output)

Returns:

This method return one of the possible value from s_ret

See also:

s_ret (p. 196)

Implemented in **CFile** (p. 43), **CPipe** (p. 57), **CSerial** (p. 66), and **CSocket** (p. 72).

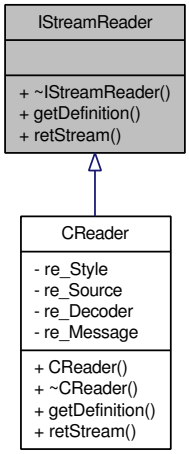
The documentation for this class was generated from the following file:

- **IStreamDaq.h**

10.22 IStreamReader Class Reference

Interface for different type of stream reader.

Inheritance diagram for IStreamReader:



Public Member Functions

- virtual **s_ret getDefinition** (struct **SDef** def)=0
- virtual **s_ret retStream** (std::vector< struct **SMessage** > *buffer)=0

10.22.1 Constructor & Destructor Documentation

10.22.1.1 virtual IStreamReader::~IStreamReader () [virtual]

10.22.2 Member Function Documentation

10.22.2.1 virtual s_ret IStreamReader::getDefinition (struct SDef *def*) [pure virtual]

Initialize all the classes linked with reader with the stream definition.

Returns:

This method return one of the possible value from s_ret

See also:

s_ret (p. 196)

Implemented in **CReader** (p. 62).

10.22.2.2 virtual s_ret IStreamReader::retStream (std::vector< struct SMessage > * *buffer*) [pure virtual]

This method divides the packet recived into message to be formatted (the lenght of each message is the pkt_lenght defined in the xml file).

Parameters:

buffer Pointer to a pointer of a vector of the messages to be formatted (output)

Returns:

This method return one of the possible value from s_ret

See also:

s_ret (p. 196)

Implemented in **CReader** (p. 62).

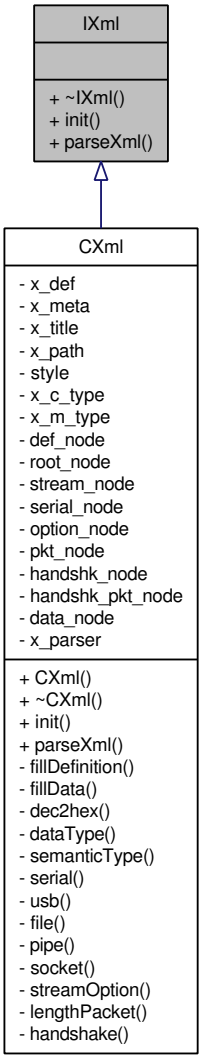
The documentation for this class was generated from the following file:

- **IStreamReader.h**

10.23 IXml Class Reference

Interface for xml parser.

Inheritance diagram for IXml:



Public Member Functions

- virtual **x_ret init** ()=0
- virtual **x_ret parseXml** ()=0

10.23.1 Constructor & Destructor Documentation

10.23.1.1 virtual IXml::~IXml () [virtual]

10.23.2 Member Function Documentation

10.23.2.1 virtual x_ret IXml::init () [pure virtual]

Initialize the parser.

Returns:

This method return one of the possible value from enum

See also:

x_ret (p. 199)

Implemented in **CXml** (p. 79).

10.23.2.2 virtual x_ret IXml::parseXml () [pure virtual]

This method make the parsing operation and fill def and metadata (validating the parsed value).

Parameters:

title This is a pointer to a variable contains the title of the output vector (output)

Returns:

This method return one of the possible value from enum

See also:

x_ret (p. 199)

Implemented in **CXml** (p. 79).

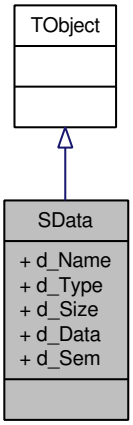
The documentation for this class was generated from the following file:

- **IXml.h**

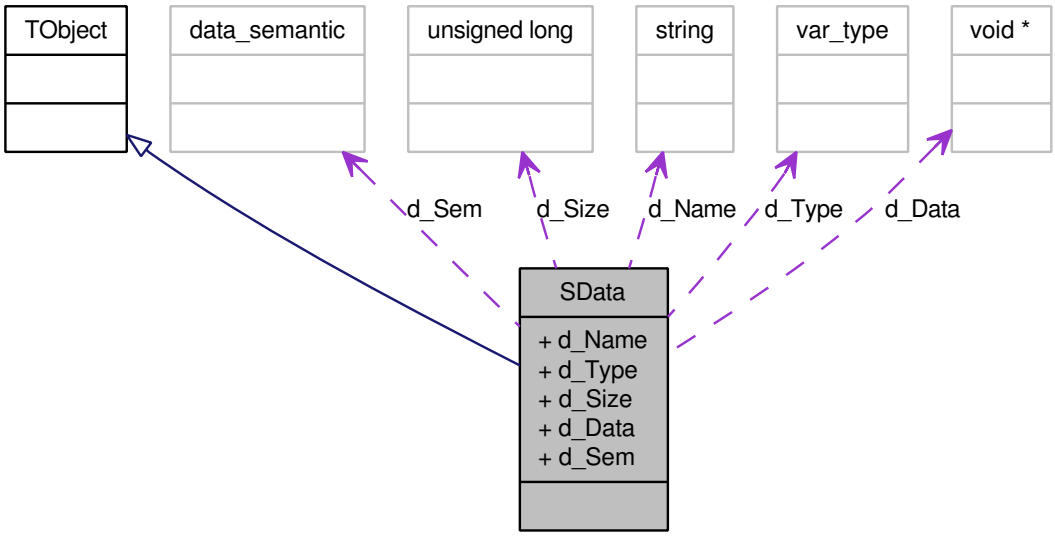
10.24 SData Class Reference

Structure of data elements.

Inheritance diagram for SData:



Collaboration diagram for SData:



Public Attributes

- std::string **d_Name**
- **var_type d_Type**
- unsigned long **d_Size**
- void * **d_Data**
- **data_semantic d_Sem**

10.24.1 Member Data Documentation

10.24.1.1 std::string SData::d_Name

Element name

10.24.1.2 var_type SData::d_Type

Type of the element

10.24.1.3 unsigned long SData::d_Size

Size of a d_Type vector (in numeber of d_type element)

10.24.1.4 void* SData::d_Data

Pointer to a data

10.24.1.5 data_semantic SData::d_Sem

Data semantic value

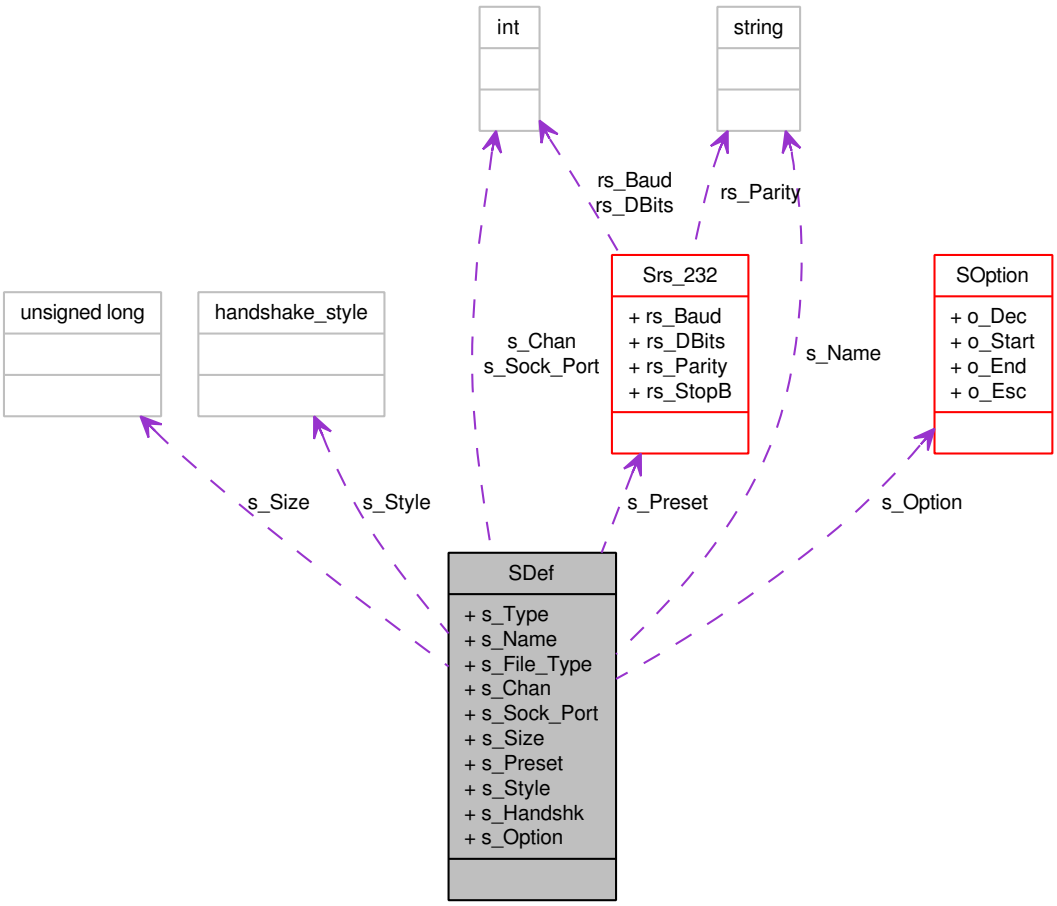
The documentation for this class was generated from the following file:

- **Data_struct.h**

10.25 SDef Struct Reference

Structure for stream definition.

Collaboration diagram for SDef:



Public Attributes

- **stream_type s_Type**
- std::string **s_Name**
- **file_type s_File_Type**
- unsigned int **s_Chan**
- unsigned int **s_Sock_Port**
- unsigned long **s_Size**
- struct **Srs_232** * **s_Preset**
- **handshake_style s_Style**
- std::vector< struct **SHandshaking** > * **s_Handshk**
- struct **SOption** * **s_Option**

10.25.1 Member Data Documentation

10.25.1.1 stream_type SDef::s_Type

Stream type

10.25.1.2 std::string SDef::s_Name

Stream name

10.25.1.3 file_type SDef::s_File_Type

File type (only for file type)

10.25.1.4 unsigned int SDef::s_Chan

Logical channel number (from stream_info attribute in xml)

10.25.1.5 unsigned int SDef::s_Sock_Port

Socket port number (from stream_info attribute in xml)

10.25.1.6 unsigned long SDef::s_Size

Packet total size

10.25.1.7 struct Srs_232* SDef::s_Preset [read]

RS-232 presets (baud rate, number of data bit, ...)

See also:

Srs_232 (p. 125)

10.25.1.8 handshake_style SDef::s_Style

Handshake style

See also:

handshake_style (p. 156)

10.25.1.9 std::vector<struct SHandshaking>* SDef::s_Handshk

Handshaking packets to start communication

See also:

SHandshaking (p. 119)

10.25.1.10 struct SOption* SDef::s_Option [read]

Character of syncro and of escape

See also:

SOption (p. 124)

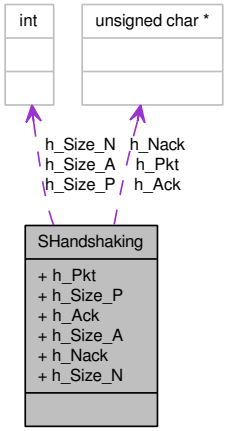
The documentation for this struct was generated from the following file:

- **DAQ_struct.h**

10.26 SHandshaking Struct Reference

Structure of handshaking packets.

Collaboration diagram for SHandshaking:



Public Attributes

- unsigned char * **h_Pkt**
- unsigned int **h_Size_P**
- unsigned char * **h_Ack**
- unsigned int **h_Size_A**
- unsigned char * **h_Nack**
- unsigned int **h_Size_N**

10.26.1 Member Data Documentation

10.26.1.1 unsigned char* SHandshaking::h_Pkt

Packet to send

10.26.1.2 unsigned int SHandshaking::h_Size_P

Packet to send size

10.26.1.3 unsigned char* SHandshaking::h_Ack

Ack value

10.26.1.4 unsigned int SHandshaking::h_Size_A

Ack size

10.26.1.5 unsigned char* SHandshaking::h_Nack

Nack value

10.26.1.6 unsigned int SHandshaking::h_Size_N

Nack size

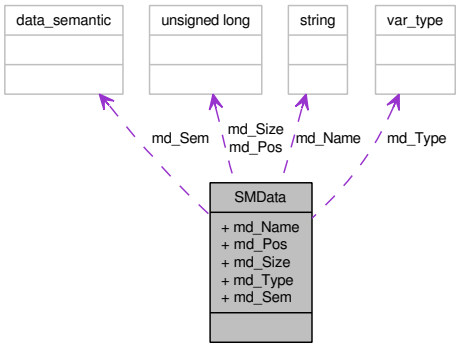
The documentation for this struct was generated from the following file:

- **DAQ_struct.h**

10.27 SMData Struct Reference

Structure of metadata elements.

Collaboration diagram for SMData:



Public Attributes

- std::string **md_Name**
- unsigned long **md_Pos**
- unsigned long **md_Size**
- **var_type md_Type**
- **data_semantic md_Sem**

10.27.1 Member Data Documentation

10.27.1.1 std::string SMData::md_Name

Element name

10.27.1.2 unsigned long SMData::md_Pos

Position of the element in the packet (in bytes)

10.27.1.3 unsigned long SMData::md_Size

Size of the element (in bytes)

10.27.1.4 var_type SMData::md_Type

Type of the element

10.27.1.5 data_semantic SMData::md_Sem

Data semantic value

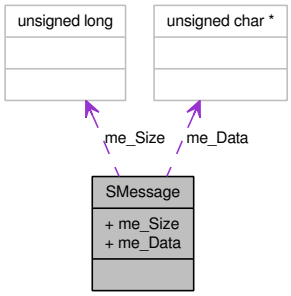
The documentation for this struct was generated from the following file:

- **DAQ_struct.h**

10.28 SMessage Struct Reference

Structure of message.

Collaboration diagram for SMessage:



Public Attributes

- unsigned long **me_Size**
- unsigned char * **me_Data**

10.28.1 Member Data Documentation

10.28.1.1 unsigned long SMessage::me_Size

Size of the message

10.28.1.2 unsigned char* SMessage::me_Data

Pointer to the message

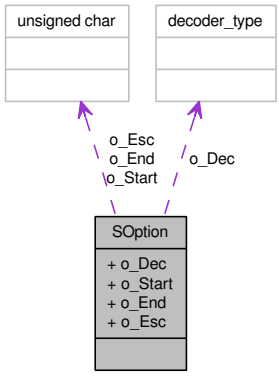
The documentation for this struct was generated from the following file:

- **DAQ_struct.h**

10.29 SOption Struct Reference

Structure of steam option: syncro character.

Collaboration diagram for SOption:



Public Attributes

- **decoder_type o_Dec**
- unsigned char **o_Start**
- unsigned char **o_End**
- unsigned char **o_Esc**

10.29.1 Member Data Documentation

10.29.1.1 decoder_type SOption::o_Dec

Decoder type

10.29.1.2 unsigned char SOption::o_Start

Packet start character

10.29.1.3 unsigned char SOption::o_End

Packet end character

10.29.1.4 unsigned char SOption::o_Esc

Escape character

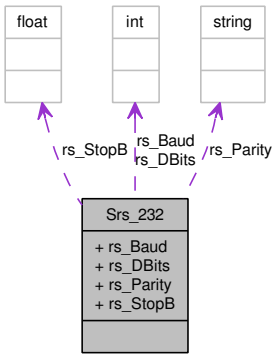
The documentation for this struct was generated from the following file:

- **DAQ_struct.h**

10.30 Srs_232 Struct Reference

Structure of RS-232 Preset.

Collaboration diagram for Srs_232:



Public Attributes

- unsigned int **rs_Baud**
- unsigned int **rs_DBits**
- std::string **rs_Parity**
- float **rs_StopB**

10.30.1 Member Data Documentation

10.30.1.1 unsigned int Srs_232::rs_Baud

Baud rate

10.30.1.2 unsigned int Srs_232::rs_DBits

Number of data bits

10.30.1.3 std::string Srs_232::rs_Parity

Parity bit

10.30.1.4 float Srs_232::rs_StopB

Number of stop bits

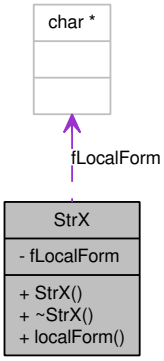
The documentation for this struct was generated from the following file:

- **DAQ_struct.h**

10.31 StrX Class Reference

This is a simple class that lets us do easy (though not terribly efficient) trancoding of XMLCh data to local code page for display.

Collaboration diagram for StrX:



Public Member Functions

- **StrX** (const XMLCh *const toTranscode)
- const char * **localForm** () const

Private Attributes

- char * **fLocalForm**

10.31.1 Constructor & Destructor Documentation

10.31.1.1 StrX::StrX (const XMLCh *const *toTranscode*)

Constructor.

Parameters:

toTranscode Data to transocode

10.31.1.2 StrX::~StrX ()

Destructor.

10.31.2 Member Function Documentation

10.31.2.1 const char* StrX::localForm () const

Getter methods.

Returns:

Data transcoded

10.31.3 Member Data Documentation

10.31.3.1 char* StrX::fLocalForm [private]

This is the local code page form of the string

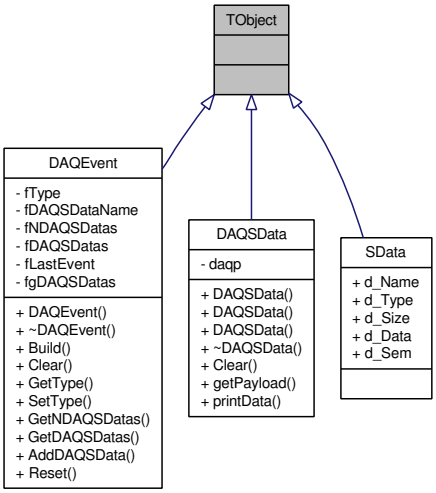
The documentation for this class was generated from the following file:

- **DOMTreeErrorReporter.hpp**

10.32 TObject Class Reference

Base class in the ROOT framework for every serializable objects.

Inheritance diagram for TObject:



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

- **DAQEvent.h**

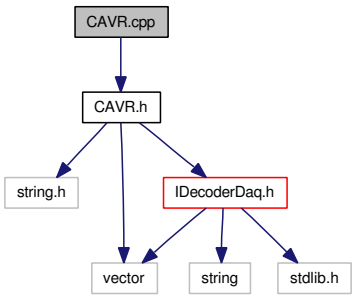
Chapter 11

File Documentation

11.1 CAVR.cpp File Reference

CAVR (p. 35) decoder development functions.

Include dependency graph for CAVR.cpp:

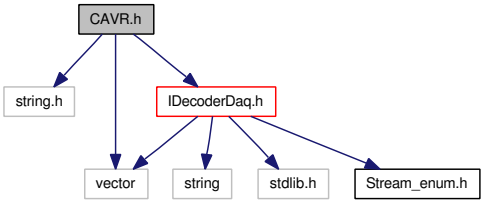


11.1.1 Detailed Description

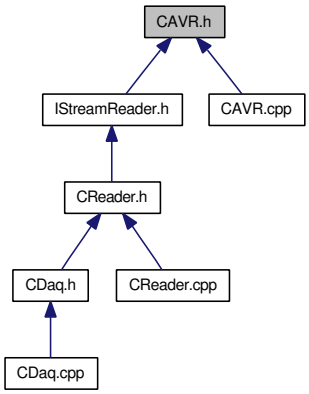
11.2 CAVR.h File Reference

Implementation of **IDecoderDaq** (p. 102) interface to decode 0x0D from AVR RS-232.

Include dependency graph for CAVR.h:



This graph shows which files directly or indirectly include this file:



Classes

- class **CAVR**

  *Implementation of **IDecoderDaq** (p. 102) interface to decode 0x0D from AVR RS-232.*

11.2.1 Detailed Description

Date:

Created on: 28-08-2009

Author:

Author: Claudio Salvadori

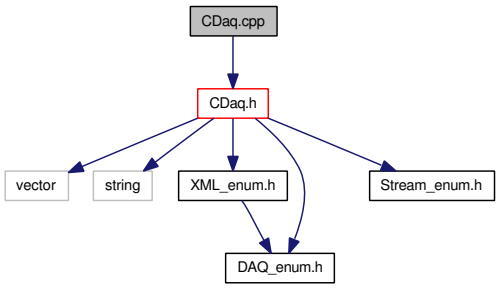
Warning:

The character 0x0A coming from RS-232 communication from AVR microcontroller is translated in 0x0D 0x0A

11.3 CDaq.cpp File Reference

CDaq (p. 38) methods development.

Include dependency graph for CDaq.cpp:



Defines

- #define **XML_PREFIX**
- #define **READER_PREFIX**
- #define **FORMATTER_PREFIX**

11.3.1 Detailed Description

11.3.2 Define Documentation

11.3.2.1 #define XML_PREFIX

Value to be added to CXmlParser return value to be translated to **CDaq** (p. 38) return value

See also:

daq_ret (p. 158)
x_ret (p. 199)

11.3.2.2 #define READER_PREFIX

Value to be added to **CReader** (p. 61) return value to be translated to **CDaq** (p. 38) return value

See also:

daq_ret (p. 158)
s_ret (p. 196)

11.3.2.3 #define FORMATTER_PREFIX

Value to be added to **CFormatter** (p. 46) return value to be translated to **CDaq** (p. 38) return value

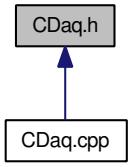
See also:

daq_ret (p. 158)
f_ret (p. 175)

11.4 CDaq.h File Reference

User interface from the data acquisition library and the user application.

This graph shows which files directly or indirectly include this file:



Classes

- class **CDaq**

  User interface from the data acquisition library and the user application.

11.4.1 Detailed Description

Date:

Created on: 31-mag-2009

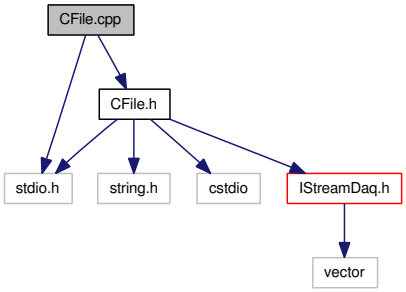
Author:

Author: Claudio Salvadori, Christian Nastasi and Paolo Pagano

11.5 CFile.cpp File Reference

File reading functions development.

Include dependency graph for CFile.cpp:

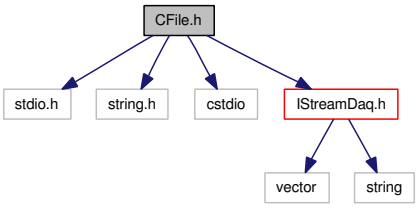


11.5.1 Detailed Description

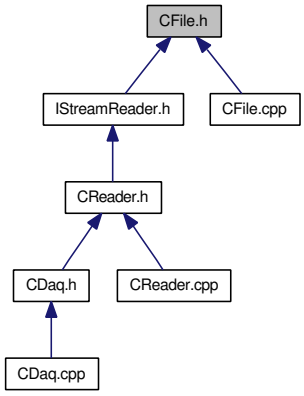
11.6 CFile.h File Reference

Implementation of IStream interface to read a file. This class reads a file and creates packet shaped like the **SDef** (p. 116) structure directives.

Include dependency graph for CFile.h:



This graph shows which files directly or indirectly include this file:



Classes

- class **CFile**

  *Implementation of IStream interface to read a file. This class reads a file and creates packet shaped like the **SDef** (p. 116) structure directives.*

11.6.1 Detailed Description

Date:

Created on: 22-apr-2009

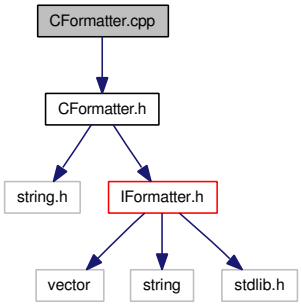
Author:

Author: Claudio Salvadori

11.7 CFormatter.cpp File Reference

Formatting functions development.

Include dependency graph for CFormatter.cpp:



11.7.1 Detailed Description

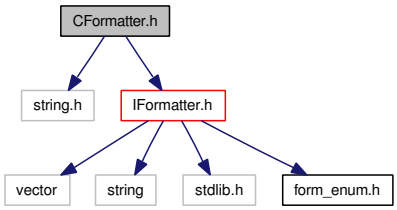
11.7.2 Define Documentation

11.7.2.1 #define U_SIZE

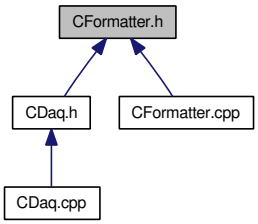
11.8 CFormatter.h File Reference

Class to format data received from stream based on the metadata vector (created by the xml parser).

Include dependency graph for CFormatter.h:



This graph shows which files directly or indirectly include this file:



Classes

- class **CFormatter**

  Class to format data received from stream based on the metadata vector (created by the xml parser).

11.8.1 Detailed Description

Date:

Created on: 22-apr-2009

Author:

Author: Claudio Salvadori

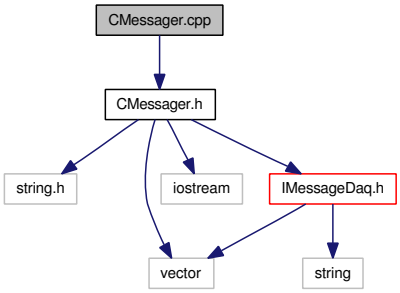
See also:

SMData (p. 121)

11.9 CMessager.cpp File Reference

Divisor in messages functions development.

Include dependency graph for CMessager.cpp:

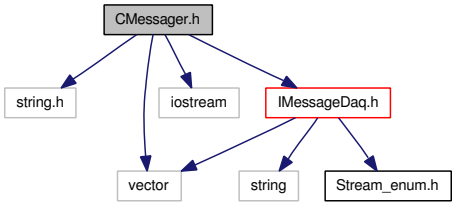


11.9.1 Detailed Description

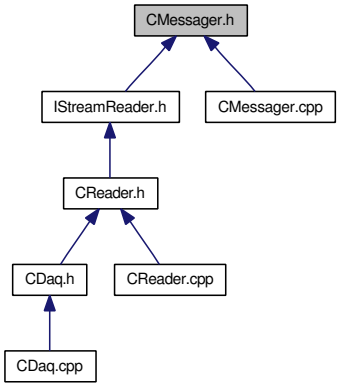
11.10 CMessager.h File Reference

Implementation of **IMessageDaq** (p. 106) interface to devide in messages a packet.

Include dependency graph for CMessager.h:



This graph shows which files directly or indirectly include this file:



Classes

- class **CMessager**

  *Implementation of **IMessageDaq** (p. 106) interface to devide in messages a packet.*

11.10.1 Detailed Description

Date:

Created on: 26-apr-2009

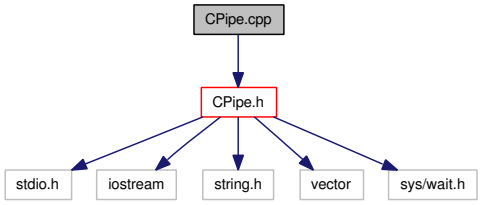
Author:

Author: Claudio Salvadori

11.11 CPipe.cpp File Reference

Pipe access functions develpoment.

Include dependency graph for CPipe.cpp:



11.11.1 Detailed Description

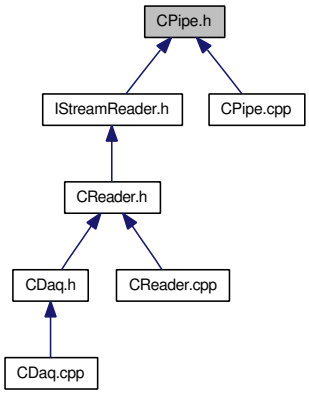
11.11.2 Define Documentation

11.11.2.1 #define MAX_ACK_LENGHT

11.12 CPipe.h File Reference

Class to read from a pipe (tread intercommunication method). Implementation of IStream interface to read a stream from a pipe. This class reads bytes from a pipe and creates packets shaped like the **SDef** (p. 116) structure directives. Daq is client of the pipe created by the server thread.

This graph shows which files directly or indirectly include this file:



Classes

- class **CPipe**

  *Class to read from a pipe (tread intercommunication method). Implementation of IStream interface to read a stream from a pipe. This class reads bytes from a pipe and creates packets shaped like the **SDef** (p. 116) structure directives. Daq is client of the pipe created by the server thread.*

11.12.1 Detailed Description

Date:

Created on: 22-apr-2009

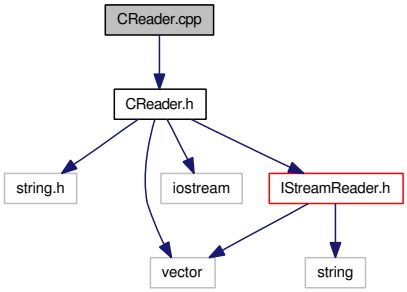
Author:

Author: Claudio Salvadori

11.13 CReader.cpp File Reference

Reader functions development.

Include dependency graph for CReader.cpp:

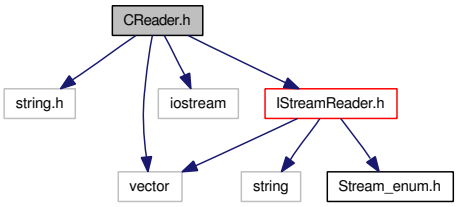


11.13.1 Detailed Description

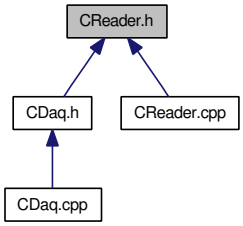
11.14 CReader.h File Reference

Implemenation for **IStreamReader** (p. 110) interface to read, decode and divide packet to be passed to formatter. This class uses instantiation of the classes **IStreamDaq** (p. 108), **IDecoderDaq** (p. 102), **IMessageDaq** (p. 106).

Include dependency graph for CReader.h:



This graph shows which files directly or indirectly include this file:



Classes

- class **CReader**

  *Implemenation for **IStreamReader** (p. 110) interface to read, decode and divide packet to be passed to formatter. This class uses instantiation of the classes **IStreamDaq** (p. 108), **IDecoderDaq** (p. 102), **IMessageDaq** (p. 106).*

11.14.1 Detailed Description

Date:

Created on: 29-apr-2009

Author:

Author: Claudio Salvadori

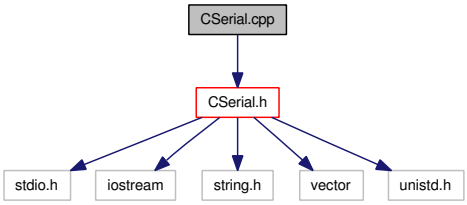
11.14.2 Define Documentation

11.14.2.1 #define CMESSAGE_H

11.15 CSerial.cpp File Reference

Serial access functions development.

Include dependency graph for CSerial.cpp:



11.15.1 Detailed Description

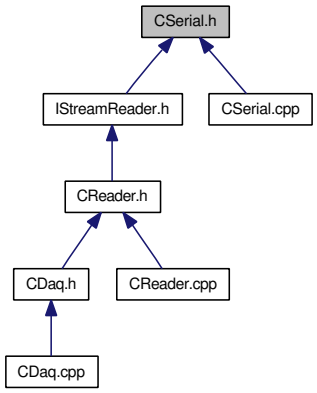
11.15.2 Define Documentation

11.15.2.1 #define MAX_ACK_LENGHT

11.16 CSerial.h File Reference

Implementation of IStream interface to read from serial (RS232). This class reads a file and creates packet shaped like the **SDef** (p. 116) structure directives.

This graph shows which files directly or indirectly include this file:



Classes

- class **CSerial**

  *Implementation of IStream interface to read from serial (RS232). This class reads a file and creates packet shaped like the **SDef** (p. 116) structure directives.*

11.16.1 Detailed Description

Date:

Created on: 27-mar-2009

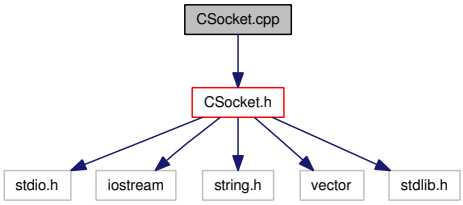
Author:

Author: Claudio Salvadori

11.17 CSocket.cpp File Reference

Socket access functions develpoment.

Include dependency graph for CSocket.cpp:



11.17.1 Detailed Description

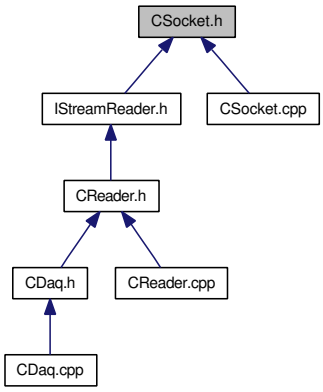
11.17.2 Define Documentation

11.17.2.1 #define MAX_ACK_LENGHT

11.18 CSocket.h File Reference

Implementation of IStream interface to read a file. This class reads a file and creates packet shaped like the **SDef** (p. 116) structure directives.

This graph shows which files directly or indirectly include this file:



Classes

- class **CSocket**

  *Implementation of IStream interface to read a stream from a socket. This class bytes form socket and creates packet shaped like the **SDef** (p. 116) structure directives.*

11.18.1 Detailed Description

Date:

Created on: 22-apr-2009

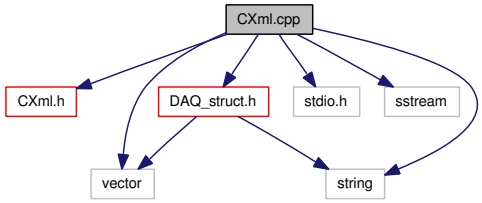
Author:

Author: Claudio Salvadori

11.19 CXml.cpp File Reference

Xml parser.

Include dependency graph for CXml.cpp:

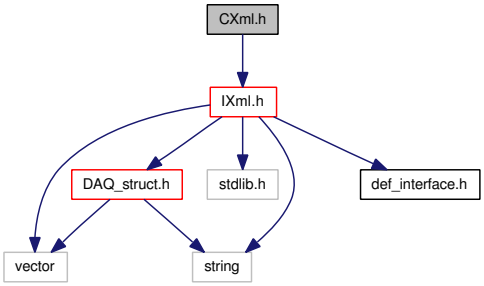


11.19.1 Detailed Description

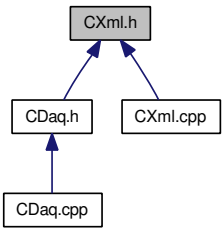
11.20 CXml.h File Reference

DAQ specific xml parsing class: class for xml parsing using xml_schema input.xsd.

Include dependency graph for CXml.h:



This graph shows which files directly or indirectly include this file:



Classes

- class **CXml**

  DAQ specific xml parsing class: class for xml parsing using xml_schema input.xsd.

11.20.1 Detailed Description

Date:

Created on: 30-mar-2009

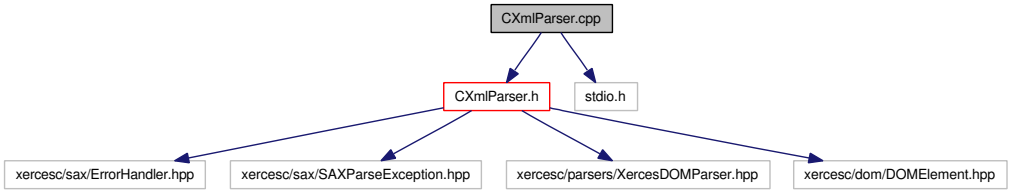
Author:

Author: Claudio Salvadori

11.21 CXmlParser.cpp File Reference

XML parsing and analysis development functions.

Include dependency graph for CXmlParser.cpp:

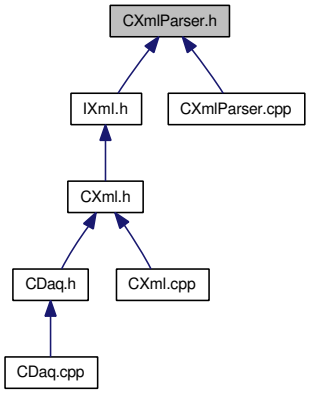


11.21.1 Detailed Description

11.22 CXmlParser.h File Reference

Class to implement methods to init, parse, process an xml file. Genercic class.

This graph shows which files directly or indirectly include this file:



Classes

- class **CXMLParser**

  Class to implement methods to init, parse, process an xml file. Genercic class.

11.22.1 Detailed Description

Date:

Created on: 23-mar-2009

Author:

Author: Claudio Salvadori

11.22.2 Define Documentation

11.22.2.1 #define strX(s)

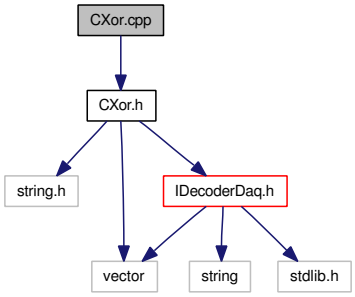
11.22.3 Variable Documentation

11.22.3.1 XERCES_CPP_NAMESPACE_USE

11.23 CXor.cpp File Reference

CXor (p. 90) decoder development functions.

Include dependency graph for CXor.cpp:

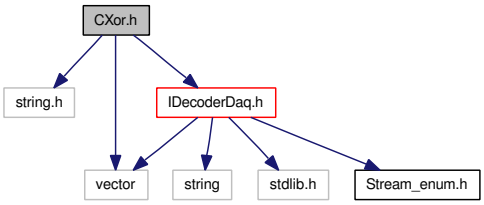


11.23.1 Detailed Description

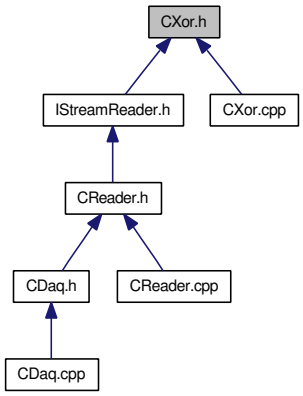
11.24 CXor.h File Reference

Implementation of **IDecoderDaq** (p. 102) interface to decode the escape character with xor function.

Include dependency graph for CXor.h:



This graph shows which files directly or indirectly include this file:



Classes

- class **CXor**

  *Implementation of **IDecoderDaq** (p. 102) interface to decode the escape character with xor function.*

11.24.1 Detailed Description

Date:

Created on: 26-apr-2009

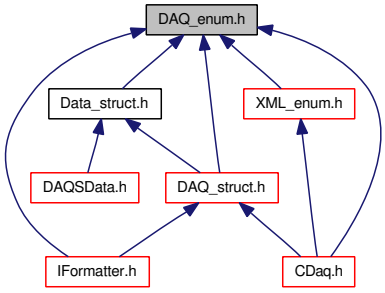
Author:

Author: Claudio Salvadori

11.25 DAQ_enum.h File Reference

Enumerations for the UI **CDaq** (p. 38) and data types definition.

This graph shows which files directly or indirectly include this file:



Defines

- #define **INT8_S**
- #define **INT16_S**
- #define **INT32_S**
- #define **INT64_S**
- #define **FLOAT32_S**
- #define **FLOAT64_S**

11.25.1 Detailed Description

Date:

Created on: 1-mag-2009

Author:

Author: Claudio Salvadori

11.25.2 Define Documentation

11.25.2.1 #define INT8_S

Definition of type size in bytes (unsigned types have the same size of signed ones).

T_INT8 and T_UINT8 size in number of byte

11.25.2.2 #define INT16_S

T_INT16 and T_UINT16 size in number of byte

11.25.2.3 #define INT32_S

T_INT32 and T_UINT32 size in number of byte

11.25.2.4 #define INT64_S

T_INT64 and T_UINT64 size in number of byte

11.25.2.5 #define FLOAT32_S

T_FLOAT32 size in number of byte

11.25.2.6 #define FLOAT64_S

T_FLOAT64 size in number of byte

11.25.3 Enumeration Type Documentation

11.25.3.1 enum decoder_type

Enumeration of available decoder for packet.

Enumerator:

D_NONE No decoder

D_XOR Decoder Xor

D_AVR Decoder AVR

11.25.3.2 enum var_type

Enumeration of possible data type defined in the xml file.

Enumerator:

T_NOTYPE Error: not correct type

T_INT8 Integer type (8bit)

T_UINT8 Integer type (8bit) unsigned

T_INT16 Integer type (16bit)

T_UINT16 Integer type (16bit) unsigned

T_INT32 Integer type (32bit)

T_UINT32 Integer type (32bit) unsigned

T_INT64 Integer type (64bit)

T_UINT64 Integer type (64bit) unsigned

T_FLOAT32 Float type (32bit)

T_FLOAT64 Float type (64bit)

11.25.3.3 enum stream_type

Enumeration of possible stream type defined in the xml file.

Enumerator:

T_SERIAL Stream from RS-232
T_USB Stream from USB
T_FILE File parsing
T_STDIN Stdin stream
T_PIPE Steam from a thread pipe
T_SOCKET Steam from a internet socket

11.25.3.4 enum file_type

Enumeration of possible file type defined in the xml file.

Enumerator:

T_TXT Text file
T_BMP Bitmap file
T_SNI Sniffer file

11.25.3.5 enum handshake_style

Enumeration of possible type of handshake in the xml file.

Enumerator:

H_S_STANDARD Standard handshake
H_S_NONE No handshake
H_S_START Handshake only for starting communication

11.25.3.6 enum data_semantic

Enumeration of the possible semantics for the data fields.

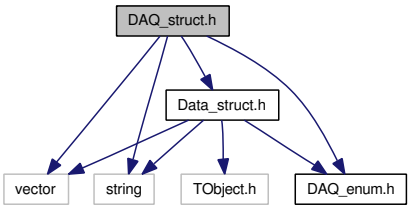
Enumerator:

DS_NOTHING Field without specific meaning (could be for eg. a controll char)
DS_TABLE Field to be inserted in a table
DS_IMAGE Field that represent a raw bitmap
DS_HISTOGRAM Field to be inserted in a histogram

11.26 DAQ_struct.h File Reference

General structures used by DAQ.

Include dependency graph for DAQ_struct.h:



Classes

- struct **SMData**

  Structure of metadata elements.

- struct **Srs_232**

  Structure of RS-232 Preset.

- struct **SHandshaking**

  Structure of handshaking packets.

- struct **SOption**

  Structure of steam option: syncro character.

- struct **SDef**

  Structure for stream definition.

- struct **SMessage**

  Structure of message.

11.26.1 Detailed Description

Date:

Created on: 10-mar-2009

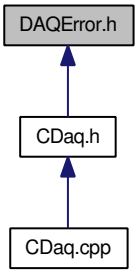
Author:

Author: Claudio Salvadori

11.27 DAQError.h File Reference

Error enumerations for the UI **CDaq** (p. 38).

This graph shows which files directly or indirectly include this file:



11.27.1 Detailed Description

Date:

Created on: 21-08-2009

Author:

Author: Claudio Salvadori

11.27.2 Enumeration Type Documentation

11.27.2.1 enum daq_ret

Enumeration of possible result **CDaq** (p. 38) class method.

Enumerator:

D_RET_OK Ok
D_INIT_ERR Error in parser init
D_PARSE_ERR Error in parsing
D_XML_PARSER_ERR Error in parsing
D_XML_P_FILE_ERR Syntax error in file.xml
D_XML_P_DOM_ERR Error in DOM parsing
D_XML_P_UN_ERR Unexpected exception during parsing
D_P_STYLE_ERR Style error
D_P_TITLE_ERR Title error
D_P_STREAM_ERR Stream node searching error
D_P_STREAM_T_ERR Stream type error
D_P_STREAM_N_ERR Stream name error
D_P_STREAM_FT_ERR Stream file type error
D_P_STREAM_C_ERR Stream channel error
D_P_PKT_ERR Packet node searching error

D_P_PKT_S_ERR Packet size error
D_P_DTA_ERR Data node searching error
D_P_DTA_N_ERR Data name error
D_P_DTA_P_ERR Data position error
D_P_DTA_S_ERR Data size error
D_P_DTA_T_ERR Data type error
D_GENERIC_ERR Generic error
D_VAL_MDATA_ERR Validate metadata error
D_P_S_STREAM_ERR Serial stream node searching error
D_P_BAUD_RATE_ERR Baud rate error
D_P_DATA_BITS_ERR Data bits error
D_P_PARITY_ERR Parity bits error
D_P_STOP_ERR Stop bits error
D_P_HAND_ERR Handshaking node searching error
D_P_HAND_P_ERR Handshaking packet node searching error
D_P_H_SEND_ERR Send packet error
D_P_O_STREAM_ERR Option stream node searching error
D_P_O_TYPE_ERR Option type error
D_P_O_START_ERR Start character error
D_P_O_END_ERR End character error
D_P_O_ESC_ERR Escape character error
D_P_H_ACK_ERR Ack packet error
D_P_H_NACK_ERR Nack packet error
D_P_H_STYLE_ERR Handshaking style error
D_OPTION_ERR Option struct doesn't fill correctly
D_D_EMPTY_ERR Empty input buffer (decode)
D_D_ESCAPE_ERR Escape character in the wrong position
D_M_EMPTY_ERR Empty input buffer (divide in messages)
D_M_LENGTH_ERR Wrong packet size (divide in messages)
D_F_OPEN_ERR Error in file opening
D_F_READ_ERR Error in file reading
D_F_R_LENGTH_ERR Error: read shortest packet in file
D_F_R_START_ERR Error in reading file: start character doesn't find
D_F_R_END_ERR Error in reading file: end character doesn't find
D_F_EOS OK end of stream for files
D_S_PRESET_MISS_ERR Error: missing the serial preset
D_S_OPEN_ERR Error in serial open
D_S_G_CHAR_ERR Error: serial get char
D_S_P_CHAR_ERR Error: serial put char
D_S_READ_ERR Error in receiving from serial
D_S_WRITE_ERR Error in sending by serial

D_S_LENGHT_ERR Error: read shortest/longest packet from serial

D_S_R_TIMEOUT_ERR Error: serial receive timeout error

D_S_T_TIMEOUT_ERR Error: serial transmit timeout error

D_S_NACK Nack received during handshking (serial)

D_S_GEN_ERR Generic error(serial)

D_P_INIT_ERR Error in pipe init

D_P_G_CHAR_ERR Error: pipe get char

D_P_P_CHAR_ERR Error: pipe put char

D_P_READ_ERR Error in receiving from pipe

D_P_WRITE_ERR Error in sending by pipe

D_P_LENGHT_ERR Error: read shortest/longest packet from pipe

D_P_R_TIMEOUT_ERR Error: pipe receive timeout error

D_P_T_TIMEOUT_ERR Error: pipe transmit timeout error

D_P_NACK Nack received during handshking (pipe)

D_P_GEN_ERR Generic error(pipe)

D_P_NOTIMPL Not implemented

D_SO_INIT_ERR Error in initializing the socket

D_SO_INIT_V_ERR Error in initializing the socket(not correct version)

D_SO_INIT_R_ERR Error in initializing the socket(resolving server)

D_SO_INIT_C_ERR Error in initializing the socket(create socket error)

D_SO_INIT_CONN_ERR Error in initializing the socket(client connection error)

D_SO_WRITE_ERR Socket write error

D_SO_LENGHT_ERR Error: read shortest/longest packet from socket

D_SO_NACK Socket: Nack received during handshking

D_SO_GEN_ERR Socket: Generic error

D_SO_READ_ERR Socket read error

D_SO_CLOSED_ERR Socket connection closed

D_META_ERR Metadata vector is empty

D_H_NULL_ERR Header vector is pointed by a NULL pointer

D_H_EMPTY_ERR Header vector is empty

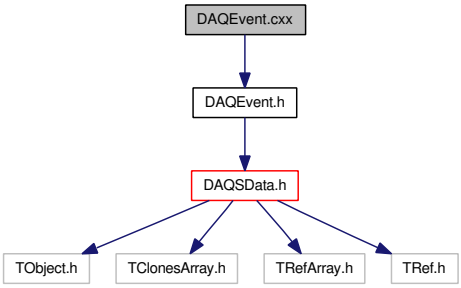
D_D_NULL_ERR Data vector is pointed by a NULL pointer

D_F_EMPTY_ERR Data vector is empty

11.28 DAQEvent.cxx File Reference

DAQEvent (p. 92) functions development.

Include dependency graph for DAQEvent.cxx:



11.28.1 Detailed Description

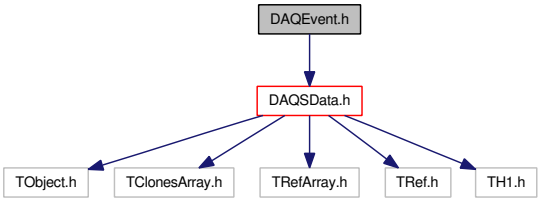
Author:

Paolo Pagano

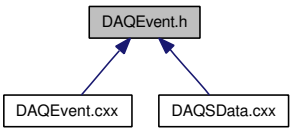
11.29 DAQEvent.h File Reference

This file contains the class to acquire an event through the link.

Include dependency graph for DAQEvent.h:



This graph shows which files directly or indirectly include this file:



Classes

- class **DAQEvent**

  Class containing members and methods for a complete event acquired through the link.

11.29.1 Detailed Description

Author:

Paolo Pagano

Date:

Wed Jul 8 11:06:26 2009

11.30 DAQEventLinkDef.h File Reference

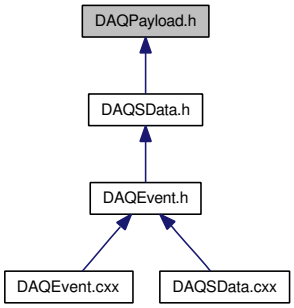
Compiling directive to generate the dictionary.

11.30.1 Detailed Description

11.31 DAQPayload.h File Reference

This file contains the stucture to store the data according with xml file.

This graph shows which files directly or indirectly include this file:



Classes

- struct **DAQPayload**

  Structure that describe the packet payload according with xml file.

11.31.1 Detailed Description

Author:

Paolo Pagano and Claudio Salvadori

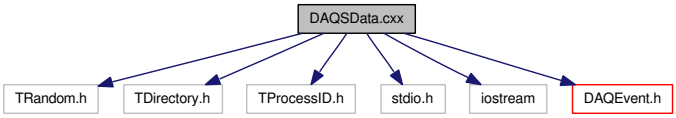
Date:

22-08-2009

11.32 DAQSData.cxx File Reference

DAQSData (p. 96) functions development.

Include dependency graph for DAQSData.cxx:



11.32.1 Detailed Description

Author:

Paolo Pagano

11.32.2 Function Documentation

11.32.2.1 void initRenderer (void)

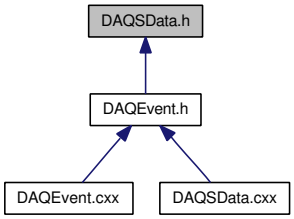
11.32.2.2 void renderImage (unsigned char * *image*, unsigned int *w*, unsigned int *h*, unsigned int *channel*)

11.32.2.3 ClassImp (DAQSData)

11.33 DAQSData.h File Reference

This file contains information on the DAQ classes.

This graph shows which files directly or indirectly include this file:



Classes

- class **DAQSData**

  Class containing members and methods for the data acquired through the link.

11.33.1 Detailed Description

Author:

Paolo Pagano and Claudio Salvadori

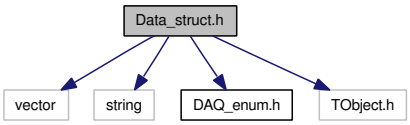
Date:

22-08-2009

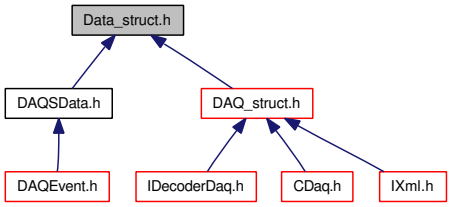
11.34 Data_struct.h File Reference

Structure of formatted data: data formatted are stored inside **SData** (p. 114) vectors.

Include dependency graph for Data_struct.h:



This graph shows which files directly or indirectly include this file:



Classes

- class **SData**

  Structure of data elements.

11.34.1 Detailed Description

Date:

Created on: 05-apr-2009

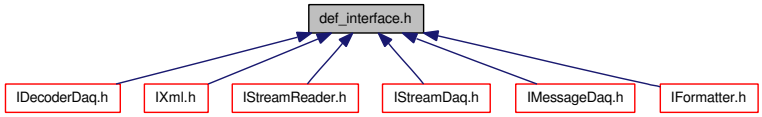
Author:

Author: Claudio Salvadori

11.35 def_interface.h File Reference

Header file to define macro for interfaces.

This graph shows which files directly or indirectly include this file:



11.35.1 Detailed Description

Date:

Created on: 23-mar-2009

Author:

Author: Claudio Salvadori

11.35.2 Define Documentation

11.35.2.1 #define Interface

11.35.2.2 #define DeclareInterface(name)

11.35.2.3 #define EndInterface

11.35.2.4 #define implements

11.36 documentation.h File Reference

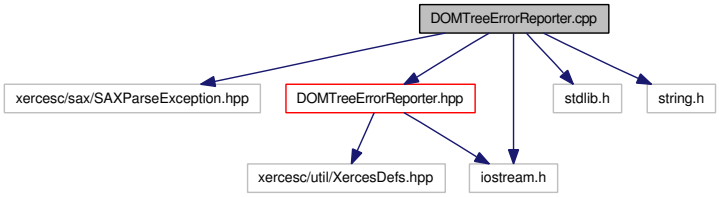
Documentation pages.

11.36.1 Detailed Description

11.37 DOMTreeErrorReporter.cpp File Reference

DOMTreeErrorReporter (p. 98) functions development.

Include dependency graph for DOMTreeErrorReporter.cpp:

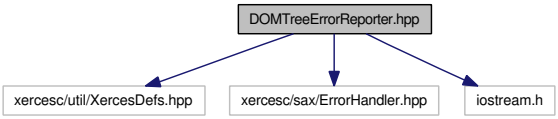


11.37.1 Detailed Description

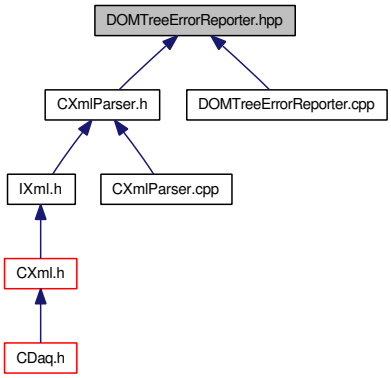
11.38 DOMTreeErrorReporter.hpp File Reference

Development of the classes **DOMTreeErrorReporter** (p. 98) and **StrX** (p. 126).

Include dependency graph for DOMTreeErrorReporter.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **DOMTreeErrorReporter**

  *This class registers as an **ErrorHandler** (p. 101) with the DOM parser and reports errors to the application.*

- class **StrX**

  This is a simple class that lets us do easy (though not terribly efficient) trancoding of XMLCh data to local code page for display.

11.38.1 Detailed Description

Date:

Created on:2006-11-06

Author:

Author: Apache Software Foundation (ASF), documented by Claudio Salvadori

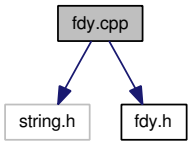
11.38.2 Function Documentation

11.38.2.1 XERCES_STD_QUALIFIER ostream& operator<< (XERCES_STD_QUALIFIER ostream & *target*, const StrX & *toDump*)

11.39 fdy.cpp File Reference

Vision algorithms development functions.

Include dependency graph for fdy.cpp:



Functions

- float **fdy_cd** (unsigned char *image, unsigned long size)

11.39.1 Detailed Description

11.39.2 Define Documentation

11.39.2.1 #define GRAY_LEVEL

11.39.3 Function Documentation

11.39.3.1 void create_histogram (unsigned char *, long unsigned *int*)

11.39.3.2 int minimum (unsigned int *a*, unsigned int *b*)

11.39.3.3 float fdy_cd (unsigned char * *image*, unsigned long *size*)

Parameters:

image Pointer to the actual image

size Image size in bytes

Returns:

Returns the percentage of similarity

11.39.3.4 void create_histogram (unsigned char * *image*, unsigned long *size*)

11.39.4 Variable Documentation

11.39.4.1 unsigned int h_old[256]

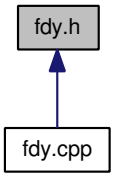
11.39.4.2 unsigned int h_act[256]

11.39.4.3 unsigned long sum

11.40 fdy.h File Reference

This function calculates the percentage of similarity using FDY algorithm.

This graph shows which files directly or indirectly include this file:



Functions

- float **fdy_cd** (unsigned char *image, unsigned long size)

11.40.1 Detailed Description

Date:

Created on: 30-apr-2009

Author:

Author: Claudio Salvadori

11.40.2 Function Documentation

11.40.2.1 float fdy_cd (unsigned char * *image*, unsigned long *size*)

Parameters:

image Pointer to the actual image

size Image size in bytes

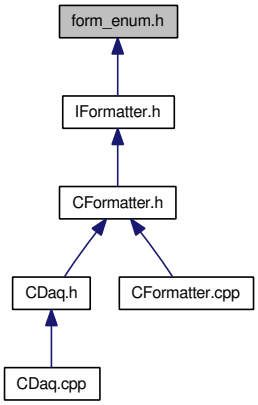
Returns:

Returns the percentage of similarity

11.41 form_enum.h File Reference

Header to enumerate the return value of **CFormatter** (p. 46) class.

This graph shows which files directly or indirectly include this file:



11.41.1 Detailed Description

Date:

Created on: 22-apr-2009

Author:

Author: Claudio Salvadori

11.41.2 Enumeration Type Documentation

11.41.2.1 enum f_ret

Enumeration of possible result **IFormatter** (p. 104) interface class method.

Enumerator:

F_RET_OK Ok

F_META_ERR Metadata vector is empty

F_H_NULL_ERR Header vector is pointed by a NULL pointer

F_H_EMPTY_ERR Header vector is empty

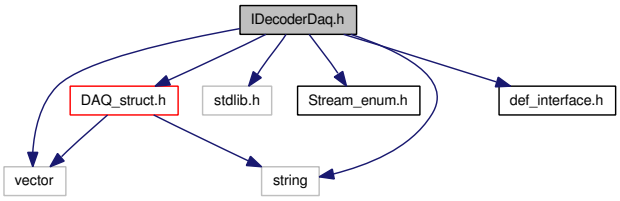
F_D_NULL_ERR Data vector is pointed by a NULL pointer

F_D_EMPTY_ERR Data vector is empty

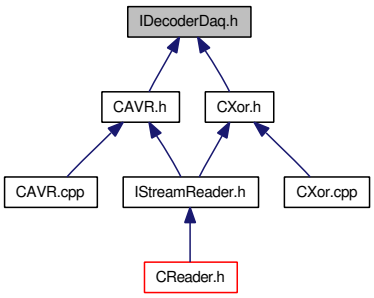
11.42 IDecoderDaq.h File Reference

Interface for the decoder classes. Decoders need to decrypt a packet received or to resolve the escape charatecter inside the packets. This class is used by the **CReader** (p. 61) class.

Include dependency graph for IDecoderDaq.h:



This graph shows which files directly or indirectly include this file:



Classes

- class **IDecoderDaq**

  *Interface for the decoder classes. Decoders need to decrypt a packet received or to resolve the escape charatecter inside the packets. This class is used by the **CReader** (p. 61) class.*

11.42.1 Detailed Description

Date:

Created on: 22-apr-2009

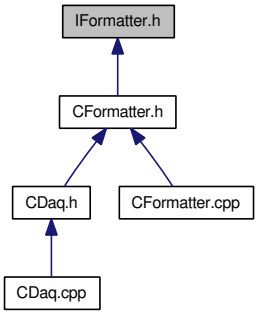
Author:

Author: Claudio Salvadori

11.43 IFormatter.h File Reference

Interface for data formatter.

This graph shows which files directly or indirectly include this file:



Classes

- class **IFormatter**

  Interface for data formatter.

11.43.1 Detailed Description

Date:

Created on: 10-mar-2009

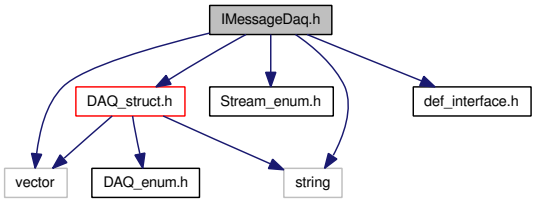
Author:

Author: Claudio Salvadori

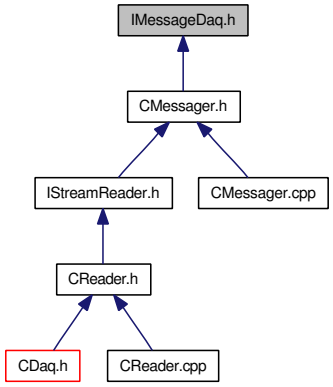
11.44 IMessageDaq.h File Reference

Interface for the messagger classes. Some application couldn't send single messages, but group of messages (for example tables): the aim of the classes those implement this inteface is to devide the data received in single messages. This class is used by the **CReader** (p. 61) class.

Include dependency graph for IMessageDaq.h:



This graph shows which files directly or indirectly include this file:



Classes

- class **IMessageDaq**

  *Interface for the messagger classes. Some application couldn't send single messages, but group of messages (for example tables): the aim of the classes those implement this inteface is to devide the data received in single messages. This class is used by the **CReader** (p. 61) class.*

11.44.1 Detailed Description

Date:

Created on: 25-apr-2009

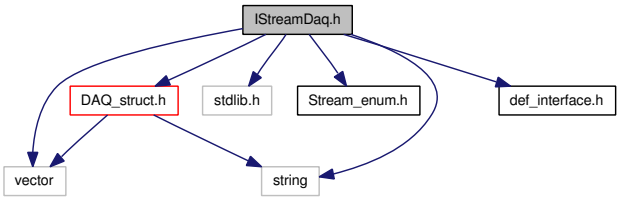
Author:

Author: Claudio Salvadori

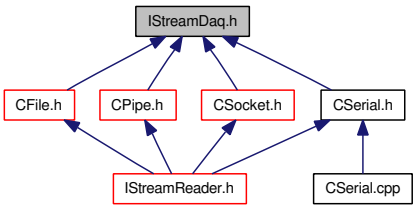
11.45 IStreamDaq.h File Reference

Interface for different type of stream. This interface is the model for the lowest layer classes that read data directly from differente devices. This class is used by the **CReader** (p. 61) class.

Include dependency graph for IStreamDaq.h:



This graph shows which files directly or indirectly include this file:



Classes

- class **IStreamDaq**

  *Interface for different type of stream. This interface is the model for the lowest layer classes that read data directly from differente devices. This class is used by the **CReader** (p. 61) class.*

11.45.1 Detailed Description

Date:

Created on: 22-apr-2009

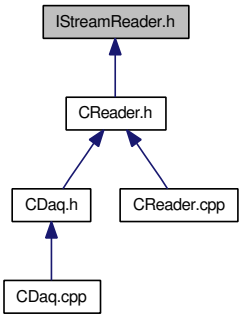
Author:

Author: Claudio Salvadori

11.46 IStreamReader.h File Reference

Interface for different type of stream reader.

This graph shows which files directly or indirectly include this file:



Classes

- class **IStreamReader**

  Interface for different type of stream reader.

11.46.1 Detailed Description

Date:

Created on: 29-apr-2009

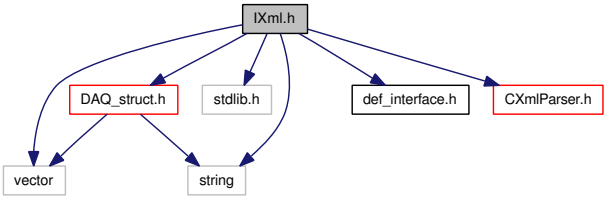
Author:

Author: Claudio Salvadori

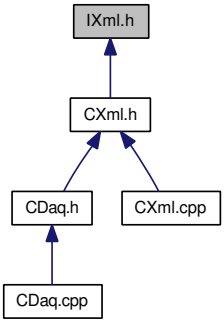
11.47 IXml.h File Reference

Interface for xml parser and analyzer.

Include dependency graph for IXml.h:



This graph shows which files directly or indirectly include this file:



Classes

- class **IXml**

  Interface for xml parser.

11.47.1 Detailed Description

Date:

Created on: 29-mar-2009

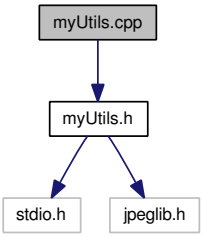
Author:

Author: Claudio Salvadori

11.48 myUtils.cpp File Reference

Vision algorithms development functions.

Include dependency graph for myUtils.cpp:



Functions

- void **my_draw_point** (unsigned char *im, unsigned int width, unsigned int height, unsigned int x, unsigned int y, **My_draw_color** color, int space, int bold)
- void **my_gray8_to_rgb24** (unsigned char *src, unsigned char *dest, unsigned int width, unsigned int height)
- void **my_rgb16_to_rgb24** (unsigned char *src, unsigned char *dest, unsigned int width, unsigned int height)
- void **my_rgb12_to_rgb24** (unsigned char *src, unsigned char *dest, unsigned int width, unsigned int height)
- void **my_gray4_to_gray8** (unsigned char *src, unsigned char *dest, unsigned int width, unsigned int height)
- void **my_gray2_to_gray8** (unsigned char *src, unsigned char *dest, unsigned int width, unsigned int height)
- void **my_bwBin_to_gray8** (unsigned char *src, unsigned char *dest, unsigned int width, unsigned int height)

11.48.1 Detailed Description

11.48.2 Typedef Documentation

11.48.2.1 typedef myJPEG_source_mgr* myJPEG_src_ptr

11.48.3 Function Documentation

11.48.3.1 void my_draw_point (unsigned char * *im*, unsigned int *width*, unsigned int *height*, unsigned int *x*, unsigned int *y*, My_draw_color *color* = MY_RED, int *space* = 6, int *bold* = 3)

Mark a point with a red cross.

Parameters:

im RGB24 image to draw

width Image width

height Image height

x Point horizontal coordinate

y Point vertical coordinate

color A My_Draw_color to use

space Size of the cross

bold Width of the lines

11.48.3.2 void my_draw_rect (unsigned char * *im*, unsigned int *width*, unsigned int *height*, unsigned int *tlx*, unsigned int *tly*, unsigned int *brx*, unsigned int *bry*, My_draw_color *color*, int *bold*)

11.48.3.3 void my_gray8_to_rgb24 (unsigned char * *src*, unsigned char * *dest*, unsigned int *width*, unsigned int *height*)

Convert from Gray8 to RGB24.

Parameters:

src Source image

dest Destination buffer capable to store the result image

width Image width

height Image height

11.48.3.4 void my_rgb16_to_rgb24 (unsigned char * *src*, unsigned char * *dest*, unsigned int *width*, unsigned int *height*)

Convert from RGB16 to RGB24.

Parameters:

src Source image

dest Destination buffer capable to store the result image

width Image width

height Image height

11.48.3.5 void my_rgb12_to_rgb24 (unsigned char * *src*, unsigned char * *dest*, unsigned int *width*, unsigned int *height*)

Convert from RGB12 to RGB24.

Parameters:

src Source image

dest Destination buffer capable to store the result image

width Image width

height Image height

11.48.3.6 void my_gray4_to_gray8 (unsigned char * *src*, unsigned char * *dest*, unsigned int *width*, unsigned int *height*)

Convert from Gray4 to Gray8.

Parameters:

src Source image

dest Destination buffer capable to store the result image

width Image width

height Image height

11.48.3.7 void my_gray2_to_gray8 (unsigned char * *src*, unsigned char * *dest*, unsigned int *width*, unsigned int *height*)

Convert from Gray2 to Gray8.

Parameters:

src Source image

dest Destination buffer capable to store the result image

width Image width

height Image height

11.48.3.8 void my_bwBin_to_gray8 (unsigned char * *src*, unsigned char * *dest*, unsigned int *width*, unsigned int *height*)

Convert from Black Binary to Gray8.

Parameters:

src Source image

dest Destination buffer capable to store the result image

width Image width

height Image height

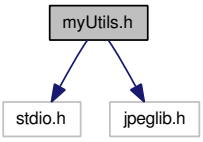
11.48.3.9 METHODDEF (void)

11.48.3.10 METHODDEF (boolean)

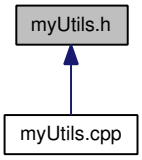
11.49 myUtils.h File Reference

Conversion functions. This is library usefull for image compression and image conversion.

Include dependency graph for myUtils.h:



This graph shows which files directly or indirectly include this file:



Functions

- bool **my_JPEG_decompress** (unsigned char *srcBuffer, int size, unsigned char *dstBuffer, int &width, int &height, int &bpp)
- void **my_rgb16_to_rgb24** (unsigned char *src, unsigned char *dest, unsigned int width, unsigned int height)
- void **my_rgb12_to_rgb24** (unsigned char *src, unsigned char *dest, unsigned int width, unsigned int height)
- void **my_gray4_to_gray8** (unsigned char *src, unsigned char *dest, unsigned int width, unsigned int height)
- void **my_gray2_to_gray8** (unsigned char *src, unsigned char *dest, unsigned int width, unsigned int height)
- void **my_gray8_to_rgb24** (unsigned char *src, unsigned char *dest, unsigned int width, unsigned int height)
- void **my_bwBin_to_gray8** (unsigned char *src, unsigned char *dest, unsigned int width, unsigned int height)
- void **my_draw_point** (unsigned char *im, unsigned int width, unsigned int height, unsigned int x, unsigned int y, **My_draw_color** color=MY_RED, int space=6, int bold=3)

11.49.1 Detailed Description

Author:

Nastasi Christian

Date:

2007-12-17

11.49.2 Enumeration Type Documentation

11.49.2.1 enum My_draw_color

Colors.

Enumerator:

MY_RED Red

MY_GREEN Green

MY_BLUE Blue

11.49.3 Function Documentation

11.49.3.1 bool my_JPEG_decompress (unsigned char * *srcBuffer*, int *size*, unsigned char * *dstBuffer*, int & *width*, int & *height*, int & *bpp*)

JPEG Decompression.

Parameters:

srcBuffer Source JPEG image

size Source JPEG image size

dstBuffer Destination buffer capable to store the decompressed image

width The decompressed image width

height The decompressed image height

bpp The decompressed image number of byte per pixel

Returns:

11.49.3.2 void my_rgb16_to_rgb24 (unsigned char * *src*, unsigned char * *dest*, unsigned int *width*, unsigned int *height*)

Convert from RGB16 to RGB24.

Parameters:

src Source image

dest Destination buffer capable to store the result image

width Image width

height Image height

11.49.3.3 void my_rgb12_to_rgb24 (unsigned char * *src*, unsigned char * *dest*, unsigned int *width*, unsigned int *height*)

Convert from RGB12 to RGB24.

Parameters:

src Source image

dest Destination buffer capable to store the result image

width Image width

height Image height

11.49.3.4 void my_gray4_to_gray8 (unsigned char * *src*, unsigned char * *dest*, unsigned int *width*, unsigned int *height*)

Convert from Gray4 to Gray8.

Parameters:

src Source image

dest Destination buffer capable to store the result image

width Image width

height Image height

11.49.3.5 void my_gray2_to_gray8 (unsigned char * *src*, unsigned char * *dest*, unsigned int *width*, unsigned int *height*)

Convert from Gray2 to Gray8.

Parameters:

src Source image

dest Destination buffer capable to store the result image

width Image width

height Image height

11.49.3.6 void my_gray8_to_rgb24 (unsigned char * *src*, unsigned char * *dest*, unsigned int *width*, unsigned int *height*)

Convert from Gray8 to RGB24.

Parameters:

src Source image

dest Destination buffer capable to store the result image

width Image width

height Image height

11.49.3.7 void my_bwBin_to_gray8 (unsigned char * *src*, unsigned char * *dest*, unsigned int *width*, unsigned int *height*)

Convert from Black Binary to Gray8.

Parameters:

src Source image

dest Destination buffer capable to store the result image

width Image width

height Image height

11.49.3.8 void my_draw_point (unsigned char * *im*, unsigned int *width*, unsigned int *height*, unsigned int *x*, unsigned int *y*, My_draw_color *color* = MY_RED, int *space* = 6, int *bold* = 3)

Mark a point with a red cross.

Parameters:

im RGB24 image to draw

width Image width

height Image height

x Point horizontal coordinate

y Point vertical coordinate

color A My_Draw_color to use

space Size of the cross

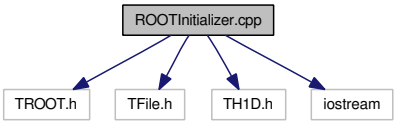
bold Width of the lines

11.49.3.9 void my_draw_rect (unsigned char * *im*, unsigned int *width*, unsigned int *height*, unsigned int *tlx*, unsigned int *tly*, unsigned int *brx*, unsigned int *bry*, My_draw_color *color* = MY_GREEN, int *bold* = 3)

11.50 ROOTInitializer.cpp File Reference

Static class to initialize the repository file.

Include dependency graph for ROOTInitializer.cpp:



11.50.1 Detailed Description

Author:

Paolo Pagano

11.50.2 Variable Documentation

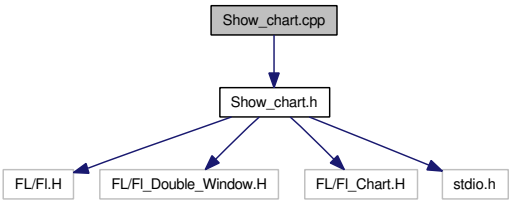
11.50.2.1 TFile* rootfile_

11.50.2.2 TH1D* myhisto_

11.51 Show_chart.cpp File Reference

Chart rendering function development.

Include dependency graph for Show_chart.cpp:



Functions

- void **initChart** (void)
- void **renderChart** (int *h, unsigned int s)

11.51.1 Detailed Description

11.51.2 Function Documentation

11.51.2.1 void initChart (void)

Function to initialize the window to draw the chart. This function has to be called before the function that execute renderImage.

See also:

renderImage (p. 193)

11.51.2.2 void renderChart (int * *h*, unsigned int *s*)

Function to draw chart.

Parameters:

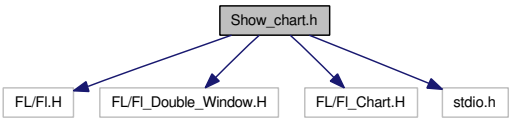
image pointer to the histogram vector

s vector size

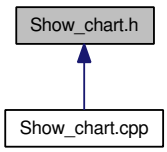
11.52 Show_chart.h File Reference

Function to draw histogram using FLTK libraries.

Include dependency graph for Show_chart.h:



This graph shows which files directly or indirectly include this file:



Functions

- void **initChart** (void)
- void **renderChart** (int *h, unsigned int s)

11.52.1 Detailed Description

Date:

Created on: 01-mag-2009

Author:

Author: Claudio Salvadori

11.52.2 Function Documentation

11.52.2.1 void initChart (void)

Function to initialize the window to draw the chart. This function has to be called before the function that execute renderImage.

See also:

renderImage (p. 193)

11.52.2.2 void renderChart (int * *h*, unsigned int *s*)

Function to draw chart.

Parameters:

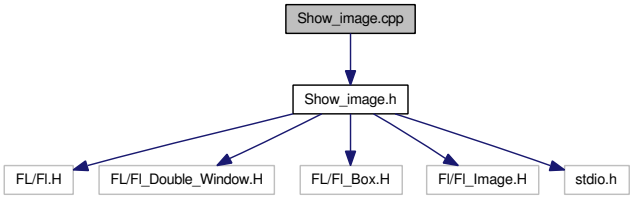
image pointer to the histogram vector

s vector size

11.53 Show_image.cpp File Reference

Image rendering function development.

Include dependency graph for Show_image.cpp:



Functions

- void **initRenderer** (void)
- void **renderImage** (unsigned char *image, unsigned int w, unsigned int h, unsigned int channel)

11.53.1 Detailed Description

11.53.2 Function Documentation

11.53.2.1 void initRenderer (void)

Function to initialize the renderer. This function has to be called before the function that execute the image rendering.

See also:

renderImage (p. 193)

11.53.2.2 void renderImage (unsigned char * *image*, unsigned int *w*, unsigned int *h*, unsigned int *channel*)

Function to rendet the image.

Parameters:

image pointer to the raw image vector

w image width

h image height

channel number of color channel

11.53.3 Variable Documentation

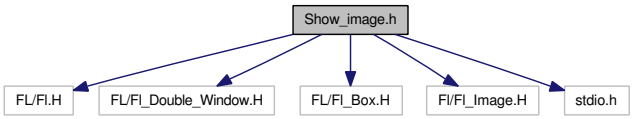
11.53.3.1 Fl_Double_Window* screen_wnd

11.53.3.2 Fl_Box* screen_img

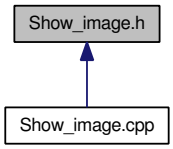
11.54 Show_image.h File Reference

Function to render a raw image using FLTK libraries.

Include dependency graph for Show_image.h:



This graph shows which files directly or indirectly include this file:



Functions

- void **initRenderer** (void)
- void **renderImage** (unsigned char *image, unsigned int w, unsigned int h, unsigned int channel)

11.54.1 Detailed Description

Date:

Created on: 01-mag-2009

Author:

Author: Claudio Salvadori

11.54.2 Function Documentation

11.54.2.1 void initRenderer (void)

Function to initialize the renderer. This function has to be called before the function that execute the image rendering.

See also:

renderImage (p. 193)

11.54.2.2 void renderImage (unsigned char * *image*, unsigned int *w*, unsigned int *h*, unsigned int *channel*)

Function to rendet the image.

Parameters:

image pointer to the raw image vector

w image width

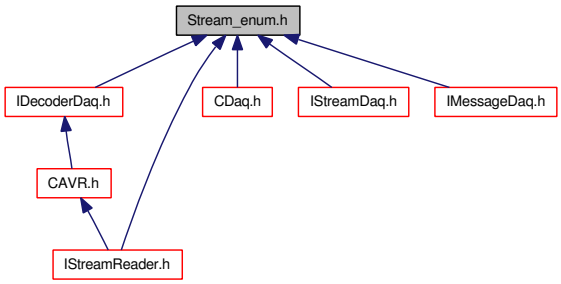
h image height

channel number of color channel

11.55 Stream_enum.h File Reference

All enumeration used by the data reader.

This graph shows which files directly or indirectly include this file:



11.55.1 Detailed Description

Date:

Created on: 29-apr-2009

Author:

Author: Claudio Salvadori

11.55.2 Enumeration Type Documentation

11.55.2.1 enum s_ret

Enumeration of possible result **IStreamReader** (p. 110) interface class method.

Enumerator:

S_RET_OK Ok

S_OPTION_ERR Option struct doesn't fill correctly

S_D_EMPTY_ERR Empty input buffer (decode)

S_D_ESCAPE_ERR Escape character in the wrong position

S_M_EMPTY_ERR Empty input buffer (divide in messages)

S_M_LENGTH_ERR Wrong packet size (divide in messages)

S_F_OPEN_ERR Error in file opening

S_F_READ_ERR Error in file reading

S_F_R_LENGTH_ERR Error: read shortest packet in file

S_F_R_START_ERR Error in reading file: start character doesn't find

S_F_R_END_ERR Error in reading file: end character doesn't find

S_F_EOS OK end of stream for files

S_S_PRESET_MISS_ERR Error: missing the serial preset

S_S_OPEN_ERR Error in serial open

S_S_G_CHAR_ERR Error: serial get char

S_S_P_CHAR_ERR Error: serial put char
S_S_READ_ERR Error in receiving from serial
S_S_WRITE_ERR Error in sending by serial
S_S_LENGHT_ERR Error: read shortest/longest packet from serial
S_S_R_TIMEOUT_ERR Error: serial receive timeout error
S_S_T_TIMEOUT_ERR Error: serial transmit timeout error
S_S_NACK Nack received during handshking (serial)
S_S_GEN_ERR Generic error(serial)
S_P_INIT_ERR Error in pipe init
S_P_G_CHAR_ERR Error: pipe get char
S_P_P_CHAR_ERR Error: pipe put char
S_P_READ_ERR Error in receiving from pipe
S_P_WRITE_ERR Error in sending by pipe
S_P_LENGHT_ERR Error: read shortest/longest packet from pipe
S_P_R_TIMEOUT_ERR Error: pipe receive timeout error
S_P_T_TIMEOUT_ERR Error: pipe transmit timeout error
S_P_NACK Nack received during handshking (pipe)
S_P_GEN_ERR Generic error(pipe)
S_P_NOTIMPL Not implemented
S_SO_INIT_ERR Error in initializing the socket
S_SO_INIT_V_ERR Error in initializing the socket(not correct version)
S_SO_INIT_R_ERR Error in initializing the socket(resolving server)
S_SO_INIT_C_ERR Error in initializing the socket(create socket error)
S_SO_INIT_CONN_ERR Error in initializing the socket(client connection error)
S_SO_WRITE_ERR Socket write error
S_SO_LENGHT_ERR Error: read shortest/longest packet from socket
S_SO_NACK Socket: Nack received during handshking
S_SO_GEN_ERR Socket: Generic error
S_SO_READ_ERR Socket read error
S_SO_CLOSED_ERR Socket connection closed

11.55.2.2 enum s_read_style

Style of reading.

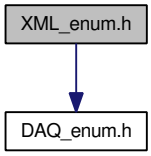
Enumerator:

S_STYLE_LENGTH Reader based on the lenght
S_STYLE_S_E Reader based on strart and end character

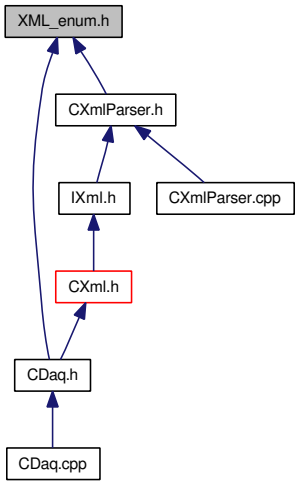
11.56 XML_enum.h File Reference

Enumeration for Xml file parsing and analysis.

Include dependency graph for XML_enum.h:



This graph shows which files directly or indirectly include this file:



11.56.1 Detailed Description

Date:

Created on: 24-apr-2009

Author:

Author: Claudio Salvadori

11.56.2 Enumeration Type Documentation

11.56.2.1 enum xml_parser_ret_value

Enumeration of possible result of xml parsing method.

Enumerator:

R_XML_OK Ok

R_XML_INIT_ERR Error in parser init

R_XML_INIT_H_ERR Error in error handler init

R_XML_PARSER_ERR Error in parsing
R_XML_P_FILE_ERR Syntax error in file.xml
R_XML_P_DOM_ERR Error in DOM parsing
R_XML_P_UN_ERR Unexpected exception during parsing

11.56.2.2 enum x_ret

Enumeration of possible result **CXml** (p. 76) class method.

Enumerator:

X_RET_OK Ok
X_INIT_ERR Error in parser init
X_PARSE_ERR Error in parsing
X_XML_PARSER_ERR Error in parsing
X_XML_P_FILE_ERR Syntax error in file.xml
X_XML_P_DOM_ERR Error in DOM parsing
X_XML_P_UN_ERR Unexpected exception during parsing
X_P_STYLE_ERR Style error
X_P_TITLE_ERR Title error
X_P_STREAM_ERR Stream node searching error
X_P_STREAM_T_ERR Stream type error
X_P_STREAM_N_ERR Stream name error
X_P_STREAM_FT_ERR Stream file type error
X_P_STREAM_C_ERR Stream channel/port error
X_P_PKT_ERR Packet node searching error
X_P_PKT_S_ERR Packet size error
X_P_DTA_ERR Data node searching error
X_P_DTA_N_ERR Data name error
X_P_DTA_P_ERR Data position error
X_P_DTA_S_ERR Data size error
X_P_DTA_T_ERR Data type error
X_GENERIC_ERR Generic error
X_VAL_MDATA_ERR Validate metadata error
X_P_S_STREAM_ERR Serial stream node searching error
X_P_BAUD_RATE_ERR Baud rate error
X_P_DATA_BITS_ERR Data bits error
X_P_PARITY_ERR Parity bits error
X_P_STOP_ERR Stop bits error
X_P_HAND_ERR Handshaking node searching error
X_P_HAND_P_ERR Handshaking packet node searching error
X_P_H_SEND_ERR Send packet error

X_P_O_STREAM_ERR Option stream node searching error
X_P_O_TYPE_ERR Option type error
X_P_O_START_ERR Start character error
X_P_O_END_ERR End character error
X_P_O_ESC_ERR Escape character error
X_P_H_ACK_ERR Ack packet error
X_P_H_NACK_ERR Nack packet error
X_P_H_STYLE_ERR Handshaking style error

11.56.2.3 enum x_style

Enumeration of possible XML file style.

Enumerator:

ST_1 Style = 1 without handshaking protocol
ST_2 Style = 2 with handshaking protocol

11.56.2.4 enum x_msg_type

Enumeration of possible type to write character.

Enumerator:

X_S_T_HEX Hexadecimal
X_S_T_CHAR Character