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

1.0

Generated by Doxygen 1.6.1

Mon Nov 16 17:32:15 2009

Contents

1 DAQ: a ROOT based Data AcQuisition platform for debugging embedded devices **1**

1.1 Motivation 1

1.2 Software architecture 1

2 Usage **5**

2.1 Installation 6

2.1.1 Cygwin issues 6

2.2 System personalization 6

2.2.1 How to add a device module? 6

2.2.2 How to add a renderer module? 7

2.2.3 System personalization 7

3 Xml-Schema **9**

4 How to fill the XML-file? **15**

4.1 Introduction 16

4.1.1 Stream title (REQUIRED) 16

4.1.2 Stream definition 16

4.1.3 Handshake definition (REQUIRED) 17

4.1.4 Payload definition 17

5 EXAMPLE : acquisition of a stream from a socket (1) **21**

5.1 How does SocketExample1.cpp work? 22

5.2 XML file example for socket acquisition 25

5.2.1 Stream title 25

5.2.2 Stream definition 25

5.2.3 Payload definition 26

5.2.4 socketLinux_1.xml 26

5.3 Payload analysis example 26

5.4 Coding intervention . . . 26
5.4.1 Payload definition . . . 27
5.4.2 DAQSData class (DAQSData.cxx) . . . 27

6 EXAMPLE : acquisition of a stream from a socket (2) 29
6.1 How does SocketExample2.cpp work? . . . 30
6.2 XML file example for socket acquisition . . . 34
6.2.1 Stream title . . . 34
6.2.2 Stream definition . . . 34
6.2.3 Handshake definition . . . 34
6.2.4 Payload definition . . . 35
6.2.5 socketLinux_2.xml . . . 35
6.3 Payload analysis example . . . 36
6.4 Coding intervention . . . 36
6.4.1 Payload definition . . . 36
6.4.2 DAQSData class (DAQSData.cxx) . . . 36

7 EXAMPLE : serial acquisition of a picture from HV7131GP and FLEX 39
7.1 XML file example for image acquisition via RS-232 . . . 40
7.1.1 Stream title . . . 40
7.1.2 Stream definition . . . 40
7.1.3 Handshake definition . . . 40
7.1.4 Payload definition . . . 41
7.1.5 serialImageLinux.xml . . . 41
7.2 Payload analysis example . . . 42
7.3 Coding intervention . . . 42
7.3.1 Payload definition . . . 42
7.3.2 DAQSData class (DAQSData.cxx) . . . 43
7.4 Post-processing images . . . 44

8 EXAMPLE : acquisition of a stream from a sniffer file 45
8.1 XML file example for sniffer acquisition . . . 46
8.1.1 Stream title . . . 46
8.1.2 Stream definition . . . 46
8.1.3 Handshake definition . . . 46
8.1.4 Payload definition . . . 46
8.1.5 XML file example for sniffer acquisition . . . 47
8.2 Payload analysis example . . . 47

8.3 Coding intervention 47
8.3.1 Payload definition 47
8.3.2 DAQSData class (DAQSData.cxx) 48

9 Todo List **49**

10 Class Index **51**
10.1 Class Hierarchy 51

11 Class Index **53**
11.1 Class List 53

12 File Index **55**
12.1 File List 55

13 Class Documentation **59**
13.1 CDaq Class Reference 59
13.1.1 Constructor & Destructor Documentation 60
13.1.1.1 CDaq 60
13.1.1.2 ∼CDaq 60
13.1.2 Member Function Documentation 60
13.1.2.1 init 60
13.1.2.2 getTitle 60
13.1.2.3 getData 61
13.1.3 Member Data Documentation 61
13.1.3.1 daq_Xml 61
13.1.3.2 daq_Reader 61
13.1.3.3 daq_Formatter 61
13.1.3.4 daq_title 61
13.1.3.5 daq_Definition 61
13.1.3.6 daq_Metadata 61
13.1.3.7 daq_Message 61
13.1.3.8 daq_Data 62
13.2 CFile Class Reference 63
13.2.1 Detailed Description 64
13.2.2 Constructor & Destructor Documentation 65
13.2.2.1 ∼CFile 65
13.2.3 Member Function Documentation 65
13.2.3.1 getDefinition 65

13.2.3.2 retStream . . . 65
13.2.3.3 File_handshaking . . . 66
13.2.3.4 File_read . . . 66
13.2.3.5 File_write . . . 66
13.2.4 Member Data Documentation . . . 66
13.2.4.1 f_Name . . . 66
13.2.4.2 f_Style . . . 67
13.2.4.3 f_Size . . . 67
13.2.4.4 f_Start . . . 67
13.2.4.5 f_End . . . 67
13.2.4.6 fdn . . . 67
13.3 CFormatter Class Reference . . . 68
13.3.1 Detailed Description . . . 70
13.3.2 Constructor & Destructor Documentation . . . 70
13.3.2.1 CFormatter . . . 70
13.3.2.2 ∼CFormatter . . . 70
13.3.3 Member Function Documentation . . . 70
13.3.3.1 init . . . 70
13.3.3.2 getData . . . 70
13.3.3.3 createDataV . . . 71
13.3.3.4 getInt32 . . . 71
13.3.3.5 getUInt32 . . . 71
13.3.3.6 getInt8 . . . 72
13.3.3.7 getUInt8 . . . 72
13.3.3.8 getInt16 . . . 72
13.3.3.9 getUInt16 . . . 72
13.3.3.10 getInt64 . . . 73
13.3.3.11 getUInt64 . . . 73
13.3.3.12 getFloat32 . . . 73
13.3.3.13 getFloat64 . . . 74
13.3.4 Member Data Documentation . . . 74
13.3.4.1 f_Data . . . 74
13.3.4.2 f_Metadata . . . 74
13.4 CGraph Class Reference . . . 75
13.4.1 Detailed Description . . . 77
13.4.2 Constructor & Destructor Documentation . . . 77

13.4.2.1 ~CGraph . . . 77
13.4.2.2 CGraph . . . 77
13.4.3 Member Function Documentation . . . 77
13.4.3.1 createObj . . . 77
13.4.3.2 setType . . . 78
13.4.3.3 render . . . 78
13.4.3.4 graphPreset . . . 78
13.4.3.5 canvasPreset . . . 78
13.4.3.6 setCanvas . . . 78
13.4.3.7 fillGraph . . . 78
13.4.4 Member Data Documentation . . . 78
13.4.4.1 gnode_ . . . 78
13.4.4.2 cnode_ . . . 79
13.4.4.3 name_ . . . 79
13.4.4.4 title_ . . . 79
13.4.4.5 max_ . . . 79
13.4.4.6 cTitle_ . . . 79
13.4.4.7 cWidth_ . . . 79
13.4.4.8 cHeight_ . . . 79
13.4.4.9 graph_ . . . 79
13.4.4.10 canv_ . . . 79
13.4.4.11 vType_ . . . 79
13.4.4.12 counter . . . 79
13.4.4.13 version . . . 79
13.4.4.14 nameGraph . . . 80
13.4.4.15 parser_ . . . 80
13.5 CHistogram Class Reference . . . 81
13.5.1 Detailed Description . . . 83
13.5.2 Constructor & Destructor Documentation . . . 83
13.5.2.1 ~CHistogram . . . 83
13.5.2.2 CHistogram . . . 83
13.5.3 Member Function Documentation . . . 83
13.5.3.1 createObj . . . 83
13.5.3.2 setType . . . 84
13.5.3.3 render . . . 84
13.5.3.4 histoPreset . . . 84

13.5.3.5 canvasPreset . . . 84
13.5.3.6 setCanvas . . . 84
13.5.3.7 fillHisto . . . 84
13.5.4 Member Data Documentation . . . 85
13.5.4.1 hnode_ . . . 85
13.5.4.2 cnode_ . . . 85
13.5.4.3 name_ . . . 85
13.5.4.4 title_ . . . 85
13.5.4.5 nBins_ . . . 85
13.5.4.6 min_ . . . 85
13.5.4.7 max_ . . . 85
13.5.4.8 cTitle_ . . . 85
13.5.4.9 cWidth_ . . . 85
13.5.4.10 cHeight_ . . . 85
13.5.4.11 histo_ . . . 85
13.5.4.12 canv_ . . . 86
13.5.4.13 vType_ . . . 86
13.5.4.14 counter . . . 86
13.5.4.15 version . . . 86
13.5.4.16 parser_ . . . 86
13.6 CImage Class Reference . . . 87
13.6.1 Detailed Description . . . 89
13.6.2 Constructor & Destructor Documentation . . . 89
13.6.2.1 ∼CImage . . . 89
13.6.2.2 CImage . . . 89
13.6.3 Member Function Documentation . . . 89
13.6.3.1 createObj . . . 89
13.6.3.2 setType . . . 89
13.6.3.3 render . . . 90
13.6.3.4 imagePreset . . . 90
13.6.4 Member Data Documentation . . . 90
13.6.4.1 inode_ . . . 90
13.6.4.2 screen_wnd . . . 90
13.6.4.3 screen_img . . . 90
13.6.4.4 vType_ . . . 90
13.6.4.5 name_ . . . 90

13.6.4.6 w_ . . . 90
13.6.4.7 h_ . . . 90
13.6.4.8 channel_ . . . 91
13.6.4.9 parser_ . . . 91
13.7 CMessager Class Reference . . . 92
13.7.1 Member Function Documentation . . . 93
13.7.1.1 getDefinition . . . 93
13.7.1.2 retStream . . . 93
13.7.2 Member Data Documentation . . . 93
13.7.2.1 msg_Size . . . 93
13.8 CNone Class Reference . . . 94
13.8.1 Constructor & Destructor Documentation . . . 95
13.8.1.1 ∼CNone . . . 95
13.8.1.2 CNone . . . 95
13.8.2 Member Function Documentation . . . 95
13.8.2.1 createObj . . . 95
13.8.2.2 render . . . 95
13.8.2.3 setType . . . 96
13.8.3 Member Data Documentation . . . 96
13.8.3.1 vType_ . . . 96
13.9 CPipe Class Reference . . . 97
13.9.1 Constructor & Destructor Documentation . . . 99
13.9.1.1 ∼CPipe . . . 99
13.9.2 Member Function Documentation . . . 99
13.9.2.1 getDefinition . . . 99
13.9.2.2 retStream . . . 99
13.9.2.3 Pipe_init . . . 99
13.9.2.4 Pipe_close . . . 100
13.9.2.5 Pipe_read . . . 100
13.9.2.6 Pipe_write . . . 100
13.9.2.7 Pipe_handshaking . . . 100
13.9.3 Member Data Documentation . . . 101
13.9.3.1 pfd . . . 101
13.9.3.2 cpid . . . 101
13.9.3.3 pi_Name . . . 101
13.9.3.4 pi_Size . . . 101

13.9.3.5 pi_Size_r . . . 101
13.9.3.6 pi_Style . . . 101
13.9.3.7 pi_HStyle . . . 101
13.9.3.8 pi_Start . . . 101
13.9.3.9 pi_End . . . 101
13.9.3.10 pi_Handshk . . . 101
13.10CReader Class Reference . . . 102
13.10.1 Constructor & Destructor Documentation . . . 103
13.10.1.1 CReader . . . 103
13.10.1.2 ∼CReader . . . 103
13.10.2 Member Function Documentation . . . 103
13.10.2.1 getDefinition . . . 103
13.10.2.2 retStream . . . 103
13.10.3 Member Data Documentation . . . 104
13.10.3.1 re_Style . . . 104
13.10.3.2 re_Source . . . 104
13.10.3.3 re_Decoder . . . 104
13.10.3.4 re_Message . . . 104
13.11CSerial Class Reference . . . 105
13.11.1 Constructor & Destructor Documentation . . . 107
13.11.1.1 ∼CSerial . . . 107
13.11.2 Member Function Documentation . . . 107
13.11.2.1 getDefinition . . . 107
13.11.2.2 retStream . . . 107
13.11.2.3 Serial_init . . . 108
13.11.2.4 Serial_close . . . 108
13.11.2.5 Serial_read . . . 108
13.11.2.6 Serial_write . . . 108
13.11.2.7 Serial_handshaking . . . 109
13.11.3 Member Data Documentation . . . 109
13.11.3.1 fd . . . 109
13.11.3.2 options . . . 109
13.11.3.3 se_Name . . . 109
13.11.3.4 se_Size . . . 109
13.11.3.5 se_Size_r . . . 109
13.11.3.6 se_Chan . . . 109

13.11.3.7 se_Style . . . 109
13.11.3.8 se_HStyle . . . 109
13.11.3.9 se_Start . . . 110
13.11.3.10 se_End . . . 110
13.11.3.11 se_Baud . . . 110
13.11.3.12 se_DBits . . . 110
13.11.3.13 se_Parity . . . 110
13.11.3.14 se_StopB . . . 110
13.11.3.15 se_Handshk . . . 110
13.12 CSocket Class Reference . . . 111
13.12.1 Constructor & Destructor Documentation . . . 113
13.12.1.1 ~CSocket . . . 113
13.12.2 Member Function Documentation . . . 113
13.12.2.1 getDefinition . . . 113
13.12.2.2 retStream . . . 113
13.12.2.3 Socket_init . . . 113
13.12.2.4 Socket_close . . . 114
13.12.2.5 Socket_read . . . 114
13.12.2.6 Socket_write . . . 114
13.12.2.7 Socket_handshaking . . . 114
13.12.3 Member Data Documentation . . . 115
13.12.3.1 so_handle . . . 115
13.12.3.2 so_Port . . . 115
13.12.3.3 so_Ip_Addr . . . 115
13.12.3.4 so_Size . . . 115
13.12.3.5 so_Size_r . . . 115
13.12.3.6 so_Style . . . 115
13.12.3.7 so_HStyle . . . 115
13.12.3.8 so_Start . . . 115
13.12.3.9 so_End . . . 115
13.12.3.10 so_Handshk . . . 116
13.13 CTable Class Reference . . . 117
13.13.1 Constructor & Destructor Documentation . . . 118
13.13.1.1 ~CTable . . . 118
13.13.1.2 CTable . . . 118
13.13.2 Member Function Documentation . . . 119

13.13.2.1 createObj . . . 119
13.13.2.2 render . . . 119
13.13.2.3 setType . . . 119
13.13.2.4 tablePreset . . . 119
13.13.3 Member Data Documentation . . . 119
13.13.3.1 tnode_ . . . 119
13.13.3.2 column_ . . . 119
13.13.3.3 name_ . . . 119
13.13.3.4 title_ . . . 119
13.13.3.5 vType_ . . . 119
13.13.3.6 file_ . . . 119
13.13.3.7 parser_ . . . 119
13.14CTarget Class Reference . . . 121
13.14.1 Constructor & Destructor Documentation . . . 123
13.14.1.1 ∼CTarget . . . 123
13.14.1.2 CTarget . . . 123
13.14.2 Member Function Documentation . . . 123
13.14.2.1 createObj . . . 123
13.14.2.2 setType . . . 123
13.14.2.3 render . . . 123
13.14.2.4 targetPreset . . . 123
13.14.3 Member Data Documentation . . . 124
13.14.3.1 tnode_ . . . 124
13.14.3.2 screen_wnd . . . 124
13.14.3.3 screen_img . . . 124
13.14.3.4 image_ . . . 124
13.14.3.5 vType_ . . . 124
13.14.3.6 name_ . . . 124
13.14.3.7 w_ . . . 124
13.14.3.8 h_ . . . 124
13.14.3.9 parser_ . . . 124
13.15CXml Class Reference . . . 125
13.15.1 Constructor & Destructor Documentation . . . 127
13.15.1.1 CXml . . . 127
13.15.1.2 ∼CXml . . . 127
13.15.2 Member Function Documentation . . . 128

13.15.2.1 init . . . 128
13.15.2.2 parseXml . . . 128
13.15.2.3 fillDefinition . . . 128
13.15.2.4 fillData . . . 128
13.15.2.5 dec2hex . . . 129
13.15.2.6 dataType . . . 129
13.15.2.7 endianessType . . . 129
13.15.2.8 serial . . . 129
13.15.2.9 usb . . . 129
13.15.2.10 file . . . 130
13.15.2.11 pipe . . . 130
13.15.2.12 socket . . . 130
13.15.2.13 streamOption . . . 130
13.15.2.14 lengthPacket . . . 130
13.15.2.15 handshake . . . 130
13.15.3 Member Data Documentation . . . 131
13.15.3.1 x_def . . . 131
13.15.3.2 x_meta . . . 131
13.15.3.3 x_title . . . 131
13.15.3.4 x_path . . . 131
13.15.3.5 x_c_type . . . 131
13.15.3.6 x_m_type . . . 131
13.15.3.7 def_node . . . 131
13.15.3.8 root_node . . . 131
13.15.3.9 stream_node . . . 131
13.15.3.10 serial_node . . . 131
13.15.3.11 option_node . . . 132
13.15.3.12 pkt_node . . . 132
13.15.3.13 handshk_node . . . 132
13.15.3.14 handshk_pkt_node . . . 132
13.15.3.15 data_node . . . 132
13.15.3.16 file_node . . . 132
13.15.3.17 x_parser . . . 132
13.16 CXMLParser Class Reference . . . 133
13.16.1 Constructor & Destructor Documentation . . . 134
13.16.1.1 CXMLParser . . . 134

13.16.1.2 ~CXMLParser . . . 134
13.16.2 Member Function Documentation . . . 134
13.16.2.1 XMLStr2CStr . . . 134
13.16.2.2 Init . . . 134
13.16.2.3 Parse . . . 135
13.16.2.4 IsTargetNode . . . 135
13.16.2.5 GetChildNode . . . 135
13.16.2.6 FindNodeRootat . . . 135
13.16.2.7 GetAttribute . . . 136
13.16.2.8 Get_NextSibling . . . 136
13.16.2.9 GetText . . . 136
13.16.2.10setTranscoder . . . 137
13.16.3 Member Data Documentation . . . 137
13.16.3.1 mParser . . . 137
13.16.3.2 mErrorHandler . . . 137
13.16.3.3 g_bTranscoder . . . 137
13.16.3.4 g_Transcoder . . . 137
13.17CXor Class Reference . . . 138
13.17.1 Member Function Documentation . . . 139
13.17.1.1 getDefinition . . . 139
13.17.1.2 retStream . . . 139
13.17.2 Member Data Documentation . . . 139
13.17.2.1 xor_Esc . . . 139
13.18DAQEvent Class Reference . . . 141
13.18.1 Constructor & Destructor Documentation . . . 143
13.18.1.1 DAQEvent . . . 143
13.18.1.2 ~DAQEvent . . . 143
13.18.2 Member Function Documentation . . . 143
13.18.2.1 Build . . . 143
13.18.2.2 Clear . . . 143
13.18.2.3 Reset . . . 143
13.18.2.4 GetType . . . 143
13.18.2.5 SetType . . . 143
13.18.2.6 GetNDAQSDatas . . . 143
13.18.2.7 GetDAQSDatas . . . 143
13.18.2.8 AddDAQSData . . . 143

13.18.3 Member Data Documentation . . . 143
13.18.3.1 fType . . . 143
13.18.3.2 fDAQSDataName . . . 143
13.18.3.3 fNDAQSDatas . . . 143
13.18.3.4 fDAQSDatas . . . 143
13.18.3.5 fLastEvent . . . 144
13.18.3.6 fgDAQSDatas . . . 144
13.19DAQPayload Struct Reference . . . 145
13.19.1 Detailed Description . . . 145
13.20DAQSData Class Reference . . . 146
13.20.1 Detailed Description . . . 147
13.20.2 Constructor & Destructor Documentation . . . 147
13.20.2.1 DAQSData . . . 147
13.20.2.2 DAQSData . . . 147
13.20.2.3 DAQSData . . . 147
13.20.2.4 ∼DAQSData . . . 147
13.20.3 Member Function Documentation . . . 147
13.20.3.1 Clear . . . 147
13.20.3.2 getPayload . . . 147
13.20.3.3 printData . . . 147
13.20.4 Member Data Documentation . . . 147
13.20.4.1 daqp . . . 147
13.21DOMTreeErrorReporter Class Reference . . . 148
13.21.1 Constructor & Destructor Documentation . . . 149
13.21.1.1 DOMTreeErrorReporter . . . 149
13.21.1.2 ∼DOMTreeErrorReporter . . . 149
13.21.2 Member Function Documentation . . . 149
13.21.2.1 warning . . . 149
13.21.2.2 error . . . 149
13.21.2.3 fatalError . . . 149
13.21.2.4 resetErrors . . . 150
13.21.2.5 getSawErrors . . . 150
13.21.3 Member Data Documentation . . . 150
13.21.3.1 fSawErrors . . . 150
13.22ErrorHandler Class Reference . . . 151
13.23IRenderer::Factory Class Reference . . . 152

13.23.1 Constructor & Destructor Documentation . . . 152
13.23.1.1 Factory . . . 152
13.23.2 Friends And Related Function Documentation . . . 152
13.23.2.1 IRenderer . . . 152
13.23.3 Member Data Documentation . . . 153
13.23.3.1 creators . . . 153
13.23.3.2 renderers . . . 153
13.23.3.3 nameV_ . . . 153
13.24IDecoderDaq Class Reference . . . 154
13.24.1 Constructor & Destructor Documentation . . . 154
13.24.1.1 ~IDecoderDaq . . . 154
13.24.2 Member Function Documentation . . . 154
13.24.2.1 getDefinition . . . 154
13.24.2.2 retStream . . . 155
13.25IFormatter Class Reference . . . 156
13.25.1 Constructor & Destructor Documentation . . . 157
13.25.1.1 ~IFormatter . . . 157
13.25.2 Member Function Documentation . . . 157
13.25.2.1 init . . . 157
13.25.2.2 getData . . . 157
13.26IMessageDaq Class Reference . . . 158
13.26.1 Constructor & Destructor Documentation . . . 158
13.26.1.1 ~IMessageDaq . . . 158
13.26.2 Member Function Documentation . . . 158
13.26.2.1 getDefinition . . . 158
13.26.2.2 retStream . . . 159
13.27IRenderer Class Reference . . . 160
13.27.1 Constructor & Destructor Documentation . . . 161
13.27.1.1 ~IRenderer . . . 161
13.27.2 Member Function Documentation . . . 161
13.27.2.1 render . . . 161
13.27.2.2 setType . . . 161
13.27.2.3 create . . . 161
13.27.2.4 getRenderer . . . 161
13.27.2.5 checkName . . . 162
13.27.2.6 endCheck . . . 162

13.28IStreamDaq Class Reference . . . 163
13.28.1 Constructor & Destructor Documentation . . . 163
13.28.1.1 ~IStreamDaq . . . 163
13.28.2 Member Function Documentation . . . 163
13.28.2.1 getDefinition . . . 163
13.28.2.2 retStream . . . 164
13.29IStreamReader Class Reference . . . 165
13.29.1 Constructor & Destructor Documentation . . . 165
13.29.1.1 ~IStreamReader . . . 165
13.29.2 Member Function Documentation . . . 165
13.29.2.1 getDefinition . . . 165
13.29.2.2 retStream . . . 166
13.30IXml Class Reference . . . 167
13.30.1 Constructor & Destructor Documentation . . . 168
13.30.1.1 ~IXml . . . 168
13.30.2 Member Function Documentation . . . 168
13.30.2.1 init . . . 168
13.30.2.2 parseXml . . . 168
13.31SData Class Reference . . . 169
13.31.1 Member Data Documentation . . . 170
13.31.1.1 d_Name . . . 170
13.31.1.2 d_Type . . . 170
13.31.1.3 d_Size . . . 170
13.31.1.4 d_Data . . . 170
13.31.1.5 d_End . . . 170
13.32SDef Struct Reference . . . 171
13.32.1 Member Data Documentation . . . 171
13.32.1.1 s_Type . . . 171
13.32.1.2 s_Name . . . 172
13.32.1.3 s_Chan . . . 172
13.32.1.4 s_Sock_Port . . . 172
13.32.1.5 s_Size . . . 172
13.32.1.6 s_Preset . . . 172
13.32.1.7 s_File . . . 172
13.32.1.8 s_Style . . . 172
13.32.1.9 s_Handshk . . . 172

13.32.1.10 s_Option . . . 173
13.33 SFile Struct Reference . . . 174
13.33.1 Member Data Documentation . . . 174
13.33.1.1 f_Header . . . 174
13.33.1.2 f_Entry_S . . . 174
13.33.1.3 f_Entry_L . . . 174
13.33.1.4 f_Trailer . . . 174
13.34 SHandshaking Struct Reference . . . 176
13.34.1 Member Data Documentation . . . 176
13.34.1.1 h_Pkt . . . 176
13.34.1.2 h_Size_P . . . 176
13.34.1.3 h_Ack . . . 176
13.34.1.4 h_Size_A . . . 177
13.34.1.5 h_Nack . . . 177
13.34.1.6 h_Size_N . . . 177
13.35 SMData Struct Reference . . . 178
13.35.1 Member Data Documentation . . . 178
13.35.1.1 md_Name . . . 178
13.35.1.2 md_Pos . . . 178
13.35.1.3 md_Size . . . 178
13.35.1.4 md_Type . . . 178
13.35.1.5 md_End . . . 179
13.36 SMessage Struct Reference . . . 180
13.36.1 Member Data Documentation . . . 180
13.36.1.1 me_Size . . . 180
13.36.1.2 me_Data . . . 180
13.37 SOption Struct Reference . . . 181
13.37.1 Member Data Documentation . . . 181
13.37.1.1 o_Dec . . . 181
13.37.1.2 o_Start . . . 181
13.37.1.3 o_End . . . 181
13.37.1.4 o_Esc . . . 181
13.38 Srs_232 Struct Reference . . . 183
13.38.1 Member Data Documentation . . . 183
13.38.1.1 rs_Baud . . . 183
13.38.1.2 rs_DBits . . . 183

13.38.1.3 rs_Parity . . . 183
13.38.1.4 rs_StopB . . . 183
13.39StrX Class Reference . . . 185
13.39.1 Constructor & Destructor Documentation . . . 185
13.39.1.1 StrX . . . 185
13.39.1.2 ~StrX . . . 185
13.39.2 Member Function Documentation . . . 186
13.39.2.1 localForm . . . 186
13.39.3 Member Data Documentation . . . 186
13.39.3.1 fLocalForm . . . 186
13.40TObject Class Reference . . . 187

14 File Documentation **189**

14.1 CDaq.cpp File Reference . . . 189
14.1.1 Detailed Description . . . 189
14.1.2 Define Documentation . . . 189
14.1.2.1 XML_PREFIX . . . 189
14.1.2.2 READER_PREFIX . . . 190
14.1.2.3 FORMATTER_PREFIX . . . 190
14.2 CDaq.h File Reference . . . 191
14.2.1 Detailed Description . . . 191
14.3 CFile.cpp File Reference . . . 192
14.3.1 Detailed Description . . . 192
14.4 CFile.h File Reference . . . 193
14.4.1 Detailed Description . . . 193
14.5 CFormatter.cpp File Reference . . . 194
14.5.1 Detailed Description . . . 194
14.5.2 Define Documentation . . . 194
14.5.2.1 U_SIZE . . . 194
14.6 CFormatter.h File Reference . . . 195
14.6.1 Detailed Description . . . 195
14.7 CGraph.cpp File Reference . . . 196
14.7.1 Detailed Description . . . 196
14.7.2 Variable Documentation . . . 196
14.7.2.1 registerFactoryG . . . 196
14.8 CGraph.h File Reference . . . 197
14.8.1 Detailed Description . . . 197

14.9 CHistogram.cpp File Reference . . . 198
14.9.1 Detailed Description . . . 198
14.9.2 Variable Documentation . . . 198
14.9.2.1 registerFactoryH . . . 198
14.10CHistogram.h File Reference . . . 199
14.10.1 Detailed Description . . . 199
14.11CImage.cpp File Reference . . . 200
14.11.1 Detailed Description . . . 200
14.11.2 Define Documentation . . . 200
14.11.2.1 EXTRA . . . 200
14.11.3 Variable Documentation . . . 200
14.11.3.1 registerFactoryI . . . 200
14.12CImage.h File Reference . . . 201
14.12.1 Detailed Description . . . 201
14.13CMessager.cpp File Reference . . . 202
14.13.1 Detailed Description . . . 202
14.14CMessager.h File Reference . . . 203
14.14.1 Detailed Description . . . 203
14.15CNone.cpp File Reference . . . 204
14.15.1 Detailed Description . . . 204
14.15.2 Variable Documentation . . . 204
14.15.2.1 registerFactoryN . . . 204
14.16CNone.h File Reference . . . 205
14.16.1 Detailed Description . . . 205
14.17CPipe.cpp File Reference . . . 206
14.17.1 Detailed Description . . . 206
14.17.2 Define Documentation . . . 206
14.17.2.1 MAX_ACK_LENGHT . . . 206
14.18CPipe.h File Reference . . . 207
14.18.1 Detailed Description . . . 207
14.19CReader.cpp File Reference . . . 208
14.19.1 Detailed Description . . . 208
14.20CReader.h File Reference . . . 209
14.20.1 Detailed Description . . . 209
14.20.2 Define Documentation . . . 209
14.20.2.1 CMESSAGE_H . . . 209

14.21CSerial.cpp File Reference 210
14.21.1 Detailed Description 210
14.21.2 Define Documentation 210
14.21.2.1 MAX_ACK_LENGHT 210
14.22CSerial.h File Reference 211
14.22.1 Detailed Description 211
14.23CSocket.cpp File Reference 212
14.23.1 Detailed Description 212
14.23.2 Define Documentation 212
14.23.2.1 MAX_ACK_LENGHT 212
14.24CSocket.h File Reference 213
14.24.1 Detailed Description 213
14.25CTable.cpp File Reference 214
14.25.1 Detailed Description 214
14.25.2 Variable Documentation 214
14.25.2.1 registerFactoryT 214
14.26CTable.h File Reference 215
14.26.1 Detailed Description 215
14.27CTarget.cpp File Reference 216
14.27.1 Detailed Description 216
14.27.2 Define Documentation 216
14.27.2.1 EXTRA 216
14.27.3 Variable Documentation 216
14.27.3.1 registerFactoryTgt 216
14.28CTarget.h File Reference 217
14.28.1 Detailed Description 217
14.29CXml.cpp File Reference 218
14.29.1 Detailed Description 218
14.30CXml.h File Reference 219
14.30.1 Detailed Description 219
14.31CXmlParser.cpp File Reference 220
14.31.1 Detailed Description 220
14.32CXmlParser.h File Reference 221
14.32.1 Detailed Description 221
14.32.2 Define Documentation 221
14.32.2.1 strX 221

14.32.3 Variable Documentation . . . 221
14.32.3.1 XERCES_CPP_NAMESPACE_USE . . . 221
14.33CXor.cpp File Reference . . . 222
14.33.1 Detailed Description . . . 222
14.34CXor.h File Reference . . . 223
14.34.1 Detailed Description . . . 223
14.35DAQ_enum.h File Reference . . . 224
14.35.1 Detailed Description . . . 224
14.35.2 Define Documentation . . . 224
14.35.2.1 INT8_S . . . 224
14.35.2.2 INT16_S . . . 224
14.35.2.3 INT32_S . . . 224
14.35.2.4 INT64_S . . . 225
14.35.2.5 FLOAT32_S . . . 225
14.35.2.6 FLOAT64_S . . . 225
14.35.3 Enumeration Type Documentation . . . 225
14.35.3.1 decoder_type . . . 225
14.35.3.2 var_type . . . 225
14.35.3.3 stream_type . . . 225
14.35.3.4 endianess_type . . . 226
14.35.3.5 handshake_style . . . 226
14.36DAQ_struct.h File Reference . . . 227
14.36.1 Detailed Description . . . 227
14.37DAQError.h File Reference . . . 228
14.37.1 Detailed Description . . . 228
14.37.2 Enumeration Type Documentation . . . 228
14.37.2.1 daq_ret . . . 228
14.38DAQEvent.cxx File Reference . . . 231
14.38.1 Detailed Description . . . 231
14.39DAQEvent.h File Reference . . . 232
14.39.1 Detailed Description . . . 232
14.40DAQEventLinkDef.h File Reference . . . 233
14.40.1 Detailed Description . . . 233
14.41DAQPayload.h File Reference . . . 234
14.41.1 Detailed Description . . . 234
14.42DAQSData.cxx File Reference . . . 235

14.42.1 Detailed Description . . . 235
14.42.2 Function Documentation . . . 235
14.42.2.1 ClassImp . . . 235
14.43DAQSData.h File Reference . . . 236
14.43.1 Detailed Description . . . 236
14.44Data_struct.h File Reference . . . 237
14.44.1 Detailed Description . . . 237
14.45documentation.h File Reference . . . 238
14.45.1 Detailed Description . . . 238
14.46DOMTreeErrorReporter.cpp File Reference . . . 239
14.46.1 Detailed Description . . . 239
14.47DOMTreeErrorReporter.hpp File Reference . . . 240
14.47.1 Detailed Description . . . 240
14.47.2 Function Documentation . . . 240
14.47.2.1 operator<< . . . 240
14.48fdy.cpp File Reference . . . 241
14.48.1 Detailed Description . . . 241
14.48.2 Define Documentation . . . 241
14.48.2.1 GRAY_LEVEL . . . 241
14.48.3 Function Documentation . . . 241
14.48.3.1 create_histogram . . . 241
14.48.3.2 minimum . . . 241
14.48.3.3 fdy_cd . . . 241
14.48.3.4 create_histogram . . . 241
14.48.4 Variable Documentation . . . 241
14.48.4.1 h_old . . . 241
14.48.4.2 h_act . . . 241
14.48.4.3 sum . . . 241
14.49fdy.h File Reference . . . 242
14.49.1 Detailed Description . . . 242
14.49.2 Function Documentation . . . 242
14.49.2.1 fdy_cd . . . 242
14.50form_enum.h File Reference . . . 243
14.50.1 Detailed Description . . . 243
14.50.2 Enumeration Type Documentation . . . 243
14.50.2.1 f_ret . . . 243

14.51globals.cxx File Reference 244
14.51.1 Detailed Description 244
14.51.2 Function Documentation 244
14.51.2.1 root 244
14.51.3 Variable Documentation 244
14.51.3.1 rootfile_ 244
14.51.3.2 event 244
14.51.3.3 tree 244
14.51.3.4 initfuncs 244
14.51.3.5 entries_ 244
14.52globals.h File Reference 245
14.52.1 Detailed Description 245
14.52.2 Function Documentation 245
14.52.2.1 InitGui 245
14.52.3 Variable Documentation 245
14.52.3.1 rootfile_ 245
14.52.3.2 event 245
14.52.3.3 tree 245
14.52.3.4 entries_ 245
14.53IDecoderDaq.h File Reference 246
14.53.1 Detailed Description 246
14.54IFormatter.h File Reference 247
14.54.1 Detailed Description 247
14.55IMessageDaq.h File Reference 248
14.55.1 Detailed Description 248
14.56IRenderer.cpp File Reference 249
14.56.1 Detailed Description 249
14.57IRenderer.h File Reference 250
14.57.1 Detailed Description 250
14.58IStreamDaq.h File Reference 251
14.58.1 Detailed Description 251
14.59IStreamReader.h File Reference 252
14.59.1 Detailed Description 252
14.60IXml.h File Reference 253
14.60.1 Detailed Description 253
14.61myUtils.cpp File Reference 254

14.61.1 Detailed Description . . . 254
14.61.2 Typedef Documentation . . . 254
14.61.2.1 myJPEG_src_ptr . . . 254
14.61.3 Function Documentation . . . 254
14.61.3.1 my_draw_point . . . 254
14.61.3.2 my_draw_rect . . . 255
14.61.3.3 my_gray8_to_rgb24 . . . 255
14.61.3.4 my_rgb16_to_rgb24 . . . 255
14.61.3.5 my_rgb12_to_rgb24 . . . 255
14.61.3.6 my_gray4_to_gray8 . . . 256
14.61.3.7 my_gray2_to_gray8 . . . 256
14.61.3.8 my_bwBin_to_gray8 . . . 256
14.61.3.9 METHODDEF . . . 256
14.61.3.10 METHODDEF . . . 256
14.62 myUtils.h File Reference . . . 257
14.62.1 Detailed Description . . . 257
14.62.2 Enumeration Type Documentation . . . 257
14.62.2.1 My_draw_color . . . 257
14.62.3 Function Documentation . . . 258
14.62.3.1 my_JPEG_decompress . . . 258
14.62.3.2 my_rgb16_to_rgb24 . . . 258
14.62.3.3 my_rgb12_to_rgb24 . . . 258
14.62.3.4 my_gray4_to_gray8 . . . 258
14.62.3.5 my_gray2_to_gray8 . . . 259
14.62.3.6 my_gray8_to_rgb24 . . . 259
14.62.3.7 my_bwBin_to_gray8 . . . 259
14.62.3.8 my_draw_point . . . 260
14.62.3.9 my_draw_rect . . . 260
14.63 ROOTInitializer.cpp File Reference . . . 261
14.63.1 Detailed Description . . . 261
14.64 Show_image.cpp File Reference . . . 262
14.64.1 Detailed Description . . . 262
14.64.2 Function Documentation . . . 262
14.64.2.1 initRenderer . . . 262
14.64.2.2 renderImage . . . 262
14.64.2.3 initRenderer_black . . . 262

14.64.2.4 renderImage_black . . . 263
14.64.2.5 initRendererSX . . . 263
14.64.2.6 renderImageSX . . . 263
14.64.2.7 initRendererDX . . . 263
14.64.2.8 renderImageDX . . . 263
14.64.2.9 initRendererDisp . . . 263
14.64.2.10 renderImageDisp . . . 263
14.64.3 Variable Documentation . . . 263
14.64.3.1 screen_wnd_SX . . . 263
14.64.3.2 screen_wnd_DX . . . 263
14.64.3.3 screen_wnd_Disp . . . 263
14.64.3.4 screen_img_SX . . . 263
14.64.3.5 screen_img_DX . . . 263
14.64.3.6 screen_img_Disp . . . 263
14.64.3.7 screen_wnd . . . 263
14.64.3.8 screen_img . . . 263
14.64.3.9 screen_wnd_black . . . 263
14.64.3.10 screen_img_black . . . 263
14.65 Show_image.h File Reference . . . 264
14.65.1 Detailed Description . . . 264
14.65.2 Function Documentation . . . 264
14.65.2.1 initRenderer . . . 264
14.65.2.2 renderImage . . . 265
14.65.2.3 initRenderer_black . . . 265
14.65.2.4 renderImage_black . . . 265
14.65.2.5 initRendererSX . . . 266
14.65.2.6 renderImageSX . . . 266
14.65.2.7 initRendererDX . . . 266
14.65.2.8 renderImageDX . . . 266
14.65.2.9 initRendererDisp . . . 266
14.65.2.10 renderImageDisp . . . 266
14.66 Stream_enum.h File Reference . . . 267
14.66.1 Detailed Description . . . 267
14.66.2 Enumeration Type Documentation . . . 267
14.66.2.1 s_ret . . . 267
14.66.2.2 s_read_style . . . 268

14.67XML_enum.h File Reference 269
14.67.1 Detailed Description 269
14.67.2 Enumeration Type Documentation 269
14.67.2.1 xml_parser_ret_value 269
14.67.2.2 x_ret 270
14.67.2.3 x_msg_type 271

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 5 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;
- **self configuring on-line data renderer** to render data following the xml presets;
- **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. 171) and **SMData** (p. 178) 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. 169)**, a structure that contains data formatted plus the information needed for the case of casting and processing.

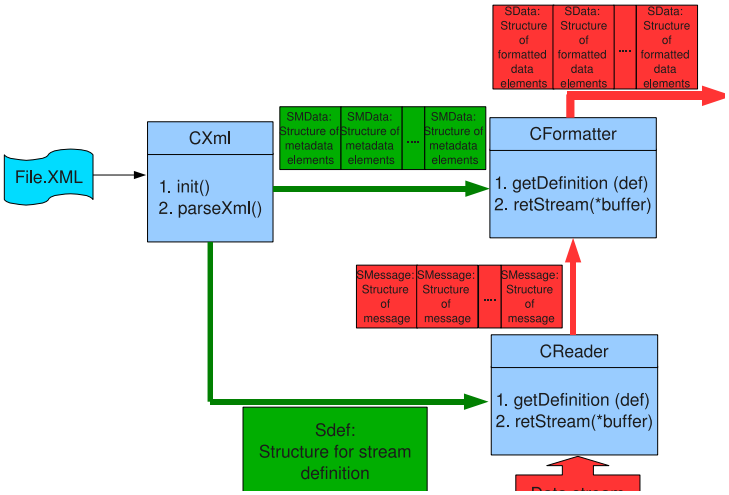


Figure 1.1: DAQ architecture

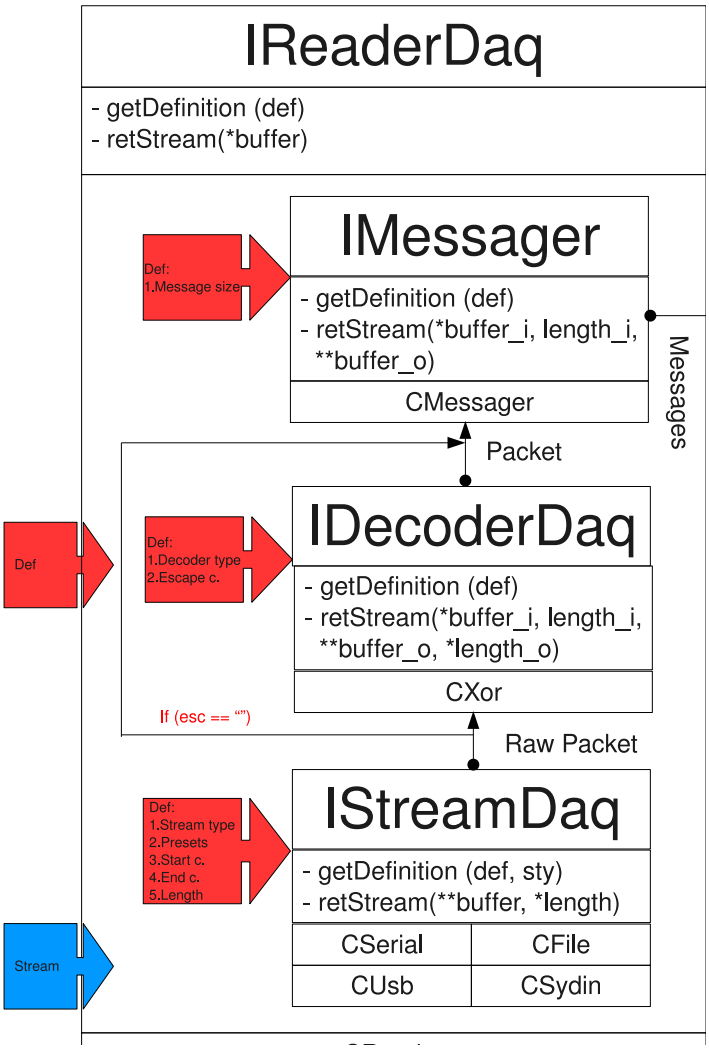


Figure 1.2: Reader structure

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. 235) and **DAQPayload.h** (p. 234) in the directory rootbulk).

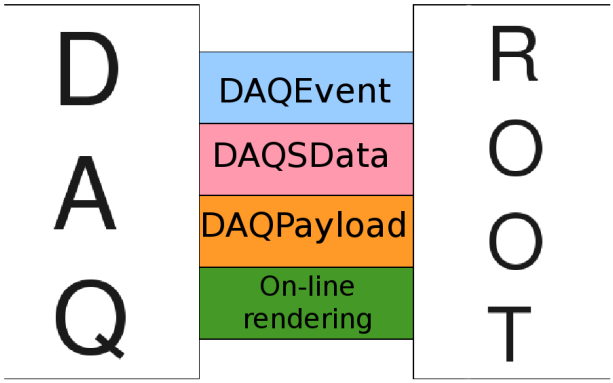


Figure 1.3: Interface DAQ-ROOT

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 device 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. 163) 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
- Update the xml parsing procedure
- Update the error enumeration
- Update the **CReader** (p. 102) class

2.2.2 How to add a renderer module?

In this subsection it is explained how to add a new data renderer class. To do this follow these steps:

- Update the **Xml-Schema** (p. 9) with the renderer presets (if they need).
- Implement the class related to renderer (inheriting from **IRenderer** (p. 160) class) and the related exception handler class.

  Warning:

  The preset parsing procedure has to be done inside the renderer class

- Register the renderer class in the object-factory

  Remarks:

  The registering procedure permits to link the renderer name defined in the xml data semantic attribute with the respective renderer class. For example, the following code shows how the image renderer is registered:

  ```
  IRenderer::Factory registerFactoryI("image", CImage::createObj);
  ```

  See also:

  CImage.cpp (p. 200)

2.2.3 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. 235) and **DAQPayload.h** (p. 234) 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="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="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="endianess_style">
 <xs:restriction base="xs:string">
  <xs:enumeration value="big-endian"/>
  <xs:enumeration value="little-endian"/>
 </xs:restriction>
</xs:simpleType>

<xs:simpleType name="channel_style">
 <xs:restriction base="xs:nonNegativeInteger">
  <xs:enumeration value="1"/>
  <xs:enumeration value="3"/>
 </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_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="header_l" type="xs:nonNegativeInteger"/>
<xs:attribute name="n_entry_s" type="xs:nonNegativeInteger"/>
<xs:attribute name="n_entry_l" type="xs:positiveInteger"/>
<xs:attribute name="trailer_s" type="xs:nonNegativeInteger"/>

<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:nonNegativeInteger"/>
<xs:attribute name="data_type" type="data_type_style"/>
<xs:attribute name="type" type="xs:string"/>
<xs:attribute name="endianess" type="endianess_style"/>

<xs:attribute name="image_name" type="xs:string"/>
<xs:attribute name="image_width" type="xs:nonNegativeInteger"/>
<xs:attribute name="image_height" type="xs:nonNegativeInteger"/>
<xs:attribute name="image_channel" type="channel_style"/>

<xs:attribute name="histogram_name" type="xs:string"/>
<xs:attribute name="histogram_title" type="xs:string"/>
<xs:attribute name="histogram_nbins" type="xs:nonNegativeInteger"/>
<xs:attribute name="histogram_min" type="xs:double"/>
<xs:attribute name="histogram_max" type="xs:double"/>

<xs:attribute name="graph_name" type="xs:string"/>
<xs:attribute name="graph_title" type="xs:string"/>
<xs:attribute name="graph_max" type="xs:double"/>

<xs:attribute name="canvas_title" type="xs:string"/>
<xs:attribute name="canvas_width" type="xs:nonNegativeInteger"/>
<xs:attribute name="canvas_height" type="xs:nonNegativeInteger"/>

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

<xs:attribute name="target_name" type="xs:string"/>
<xs:attribute name="target_width" type="xs:nonNegativeInteger"/>
<xs:attribute name="target_height" type="xs:nonNegativeInteger"/>

<!-- 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" use="required"/>
  <xs:attribute ref="data_bits" use="required"/>
  <xs:attribute ref="parity" use="required"/>
  <xs:attribute ref="stop_bits" use="required"/>
 </xs:complexType>
</xs:element>

<xs:element name="file_stream">
    <xs:complexType>
  <xs:attribute ref="header_l" use="required"/>
  <xs:attribute ref="n_entry_s" use="required"/>
  <xs:attribute ref="n_entry_l" use="required"/>
  <xs:attribute ref="trailer_s"/>
 </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="file_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_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="image_preset">
 <xs:complexType>
  <xs:attribute ref="image_name" use="required"/>
  <xs:attribute ref="image_width" use="required"/>
  <xs:attribute ref="image_height" use="required"/>
  <xs:attribute ref="image_channel" use="required"/>
 </xs:complexType>
</xs:element>

<xs:element name="histogram_preset">
 <xs:complexType>
  <xs:attribute ref="histogram_name" use="required"/>
  <xs:attribute ref="histogram_title" use="required"/>
  <xs:attribute ref="histogram_nbins" use="required"/>
  <xs:attribute ref="histogram_min" use="required"/>
  <xs:attribute ref="histogram_max" use="required"/>
 </xs:complexType>
</xs:element>

<xs:element name="graph_preset">
 <xs:complexType>
  <xs:attribute ref="graph_name" use="required"/>
  <xs:attribute ref="graph_title" use="required"/>
  <xs:attribute ref="graph_max" use="required"/>
 </xs:complexType>
</xs:element>
```

```
 <xs:element name="canvas_preset">
  <xs:complexType>
   <xs:attribute ref="canvas_title" use="required"/>
   <xs:attribute ref="canvas_width" use="required"/>
   <xs:attribute ref="canvas_height" use="required"/>
  </xs:complexType>
 </xs:element>

 <xs:element name="table_preset">
  <xs:complexType>
   <xs:attribute ref="table_name" use="required"/>
  </xs:complexType>
 </xs:element>

 <xs:element name="target_preset">
  <xs:complexType>
   <xs:attribute ref="target_name" use="required"/>
   <xs:attribute ref="target_width" use="required"/>
   <xs:attribute ref="target_height" use="required"/>
  </xs:complexType>
 </xs:element>

 <xs:element name="data">
  <xs:complexType>
   <xs:sequence>
           <xs:element ref="image_preset" minOccurs="0" maxOccurs="1"/>
           <xs:element ref="histogram_preset" minOccurs="0" maxOccurs="1"/>
           <xs:element ref="graph_preset" minOccurs="0" maxOccurs="1"/>
           <xs:element ref="canvas_preset" minOccurs="0" maxOccurs="1"/>
           <xs:element ref="table_preset" minOccurs="0" maxOccurs="1"/>
           <xs:element ref="target_preset" minOccurs="0" maxOccurs="1"/>
         </xs:sequence>
   <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:attribute ref="endianess" 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

How to fill the XML-file?

4.1 Introduction

This XML file fixes:

- the communication device settings,
- the handshake word to initialize the communication,
- the packet lenght,
- the data topology inside a packet,
- the renderer the show data.

4.1.1 Stream title (REQUIRED)

In this node is defined the title of the current data acquistion session.

4.1.2 Stream definition

All the attributes from this node and from his children nodes describe the communication device presets.

stream node (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 only ofo socket)

serial_stream node (REQUIRED for serial communication):

- baud_rate: baud rate value (allowed values: 9600, 19200, 38400, 57600, 115200) (REQUIRED)
- data_bits: number of bits for symbol (allowed values: 5, 6, 7, 8) (REQUIRED)
- parity: parity bit (allowed values: odd, even, none) (REQUIRED)
- stop_bits: numeber of stop bits (allowed values: 1, 1.5, 2) (REQUIRED)

file_stream node (REQUIRED for files):

- header_l: file header length (NOT REQUIRED)
- n_entry_s: number of payload entries start bytes (inside the header) (NOT REQUIRED)
- n_entry_l: number of payload entries length in bytes (inside the header) (NOT REQUIRED)

stream_opt node (NOT REQUIRED):

- decoder: used decoder type (allowable value are 'none' and 'xor')
- esc_char: escape character (for decoder). Could be expressed like a character or like an hexadecimal number (see s_type_opt)
- start_char: syncronization start char. Could be expressed like a character or like an hexadecimal number (see s_type_opt)
- end_char: syncronization end char. Could be expressed like a character or like an hexadecimal number (see s_type_opt)
- s_type_opt: 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.

4.1.3 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.

4.1.4 Payload definition

All the attribute from this node and from all his children nodes describe the stream payload. **packet_description** node (REQUIRED):

- length_pkt : total packet length in bytes (REQUIRED)

data node (REQUIRED):

- data_name: name of the field (REQUIRED)
- data_position: absolute position from the start of the packet in number of bytes (REQUIRED)
- data_size: size of the field in number of bytes (REQUIRED)

- data_type: data type (INT8, UINT8, INT16, UINT16, INT32, UINT32, INT64, UINT64, FLOAT32, FLOAT64) (REQUIRED)
- type: semantic of the data (nothing, table, image, histogram, graph, target) (REQUIRED). Every different renderer needs to fill the related presets nodes.
- endianess: specify the endianess of the data (big_endian, little_endian) (REQUIRED)

table_preset (REQUIRED if the data semantic is "table"):

- table_name: destination file name where data will be stored;

graph_preset (REQUIRED if the data semantic is "graph"):

- graph_name: name of the graph;
- graph_title: title of the graph;
- graph_max: maximum value for x axis.

histogram_preset (REQUIRED if the data semantic is "histogram"):

- histogram_name: name of the histogram;
- histogram_title: title of the histogram;
- histogram_nbins: number of bins;
- histogram_min: minimum value for x axis;
- histogram_max: maximum value for x axis.

canvas_preset (REQUIRED to show "graph" or "histogram"):

- canvas_title: ROOT canvas title;
- canvas_width: ROOT canvas width;
- canvas_height: ROOT canvas height.

image_preset (REQUIRED if the data semantic is "image"):

- image_name: image window name;
- image_width: image width;
- image_height: image height;
- image_channel: number of color channel.

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.

Chapter 5

EXAMPLE : acquisition of a stream from a socket (1)

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 SocketExample1.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 (this value will be saved in a file called "Counter").

5.1 How does SocketExample1.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/SocketExample1 2000).

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

Checking the correct sintax for SocketExample1 execution: port number has to be passed as parameter

```
#ifdef DAQSOCKET1
 unsigned int counter = 0;
 int sockfd, newsockfd, portno, clilen;
 char buffer[256];
 struct sockaddr_in serv_addr, cli_addr;
 int n;

 //Checking the port number
 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, escaping the char and sending packets

```
while(1){
 counter++;
 #ifdef DEBUG
 cerr << "Actual counter value: " << counter << endl;
 #endif
 unsigned int length_p = 0;
 unsigned char packet[50];

 packet[length_p++] = START;

 packet[length_p++] = (unsigned char)'D';

 packet[length_p++] = (unsigned char)'A';
```

```
packet[length_p++] = (unsigned char)'Q';

packet[length_p++] = (unsigned char)'*';

if(((unsigned char)((counter >> 24) & 0xFF)) == START){
 packet[length_p++] = ESC;
 packet[length_p++] = START ^ ESC;
}else if(((unsigned char)((counter >> 24) & 0xFF)) == END){
 packet[length_p++] = ESC;
 packet[length_p++] = packet[length_p] ^ ESC;
}else if(((unsigned char)((counter >> 24) & 0xFF)) == ESC){
 packet[length_p++] = ESC;
 packet[length_p++] = ESC ^ ESC;
}else{
 packet[length_p++] = (unsigned char)((counter >> 24) & 0xFF);
}

if(((unsigned char)((counter >> 16) & 0xFF)) == START){
 packet[length_p++] = ESC;
 packet[length_p++] = START ^ ESC;
}else if(((unsigned char)((counter >> 16) & 0xFF)) == END){
 packet[length_p++] = ESC;
 packet[length_p++] = END ^ ESC;
}else if(((unsigned char)((counter >> 16) & 0xFF)) == ESC){
 packet[length_p++] = ESC;
 packet[length_p++] = ESC ^ ESC;
}else{
 packet[length_p++] = (unsigned char)((counter >> 16) & 0xFF);
}


if(((unsigned char)((counter >> 8) & 0xFF)) == START){
 packet[length_p++] = ESC;
 packet[length_p++] = START ^ ESC;
}else if(((unsigned char)((counter >> 8) & 0xFF)) == END){
 packet[length_p++] = ESC;
 packet[length_p++] = END ^ ESC;
}else if(((unsigned char)((counter >> 8) & 0xFF)) == ESC){
 packet[length_p++] = ESC;
 packet[length_p++] = ESC ^ ESC;
}else{
 packet[length_p++] = (unsigned char)((counter >> 8) & 0xFF);
}

if(((unsigned char)((counter) & 0xFF)) == START){
 packet[length_p++] = ESC;
 packet[length_p++] = START ^ ESC;
}else if(((unsigned char)((counter) & 0xFF)) == END){
 packet[length_p++] = ESC;
 packet[length_p++] = END ^ ESC;
}else if(((unsigned char)((counter) & 0xFF)) == ESC){
 packet[length_p++] = ESC;
 packet[length_p++] = ESC ^ ESC;
}else{
 packet[length_p++] = (unsigned char)((counter) & 0xFF);
}

packet[length_p++] = END;

n = write(newsockfd, packet, length_p);
if (n < 0){
 cerr << "ERROR writing to socket" << endl;
 exit(1);
}else{
 cerr << "Packet size: " << dec << length_p << endl;
 cerr << "Packet sent: ";
```

```
   for(unsigned int k = 0; k < length_p; k++){
    cerr << hex << (int)packet[k] << ", ";
   }
   cerr << endl;
  }
 }


return 0;
```

5.2 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.

5.2.1 Stream title

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

5.2.2 Stream definition

In this case DAQ will acquire data from a socket:

- connected to the IP-address 'localhost' through the port number 2000;
- stream start character: 0x02;
- stream end character: 0x03;
- escape character: 0x20 (using xor decoder).

```
<stream stream_type="socket" stream_name="localhost" stream_info="3000">
 <stream_opt decoder="xor" start_char="02" end_char="03" esc_char="20" s_type_op
     t="hex"/>
</stream>
```

handshake Handshake definition The handshake type is 'start', the packet to send is:

- '1' (0x31)

ACK packet is:

- '1' (0x31)

and NAK packet is:

- '0' (0x30)

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

5.2.3 Payload definition

```
<packet_description length_pkt="8">
 <data data_name="Message" data_position="0" data_size="3" data_type="int8" type
     ="nothing" endianess="big-endian"/>
 <data data_name="Separator" data_position="3" data_size="1" data_type="uint8" t
     ype="nothing" endianess="big-endian"/>
 <data data_name="Counter" data_position="4" data_size="4" data_type="uint32" ty
     pe="table" endianess="big-endian">
  <table_preset table_name="Counter"/>
 </data>
</packet_description>
```

5.2.4 socketLinux_1.xml

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

 <stream stream_type="socket" stream_name="localhost" stream_info="3000">
  <stream_opt decoder="xor" start_char="02" end_char="03" esc_char="20" s_type_op
      t="hex"/>
 </stream>

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

 <packet_description length_pkt="8">
  <data data_name="Message" data_position="0" data_size="3" data_type="int8" type
      ="nothing" endianess="big-endian"/>
  <data data_name="Separator" data_position="3" data_size="1" data_type="uint8" t
      ype="nothing" endianess="big-endian"/>
  <data data_name="Counter" data_position="4" data_size="4" data_type="uint32" ty
      pe="table" endianess="big-endian">
   <table_preset table_name="Counter"/>
  </data>
 </packet_description>
</input_definition>
```

5.3 Payload analysis example

In this case the **DAQEvent** (p. 141) 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
```

5.4 Coding intervention

To configure DAQ is needed to modify the code in 2 files:

- **DAQPayload.h** (p. 234)
- **DAQSData.cxx** (p. 235)

5.4.1 Payload definition

Definition of buffer size:

```
#ifdef DAQSOCKET1
#define MESSAGE_BUFFER  4
#endif
```

Definition of the payload structure:

```
#ifdef DAQSOCKET1
 Char_t   message_[MESSAGE_BUFFER];    /**<Message arrived*/
 UChar_t   separator_;   /**<Separator value*/
 UInt_t   counter_;   /**<Counter value*/
#endif
```

5.4.2 DAQSData class (DAQSData.cxx)

Copy constructor:

```
#ifdef DAQSOCKET1
 for(int i = 0; i < MESSAGE_BUFFER; i++)
  daqp.message_[i] = orig.daqp.message_[i];

 daqp.separator_ = orig.daqp.separator_;

 daqp.counter_ = orig.daqp.counter_;
#endif
```

Default constructor:

```
#ifdef DAQSOCKET1
 for(int i = 0; i < MESSAGE_BUFFER; i++)
  daqp.message_[i] = 0;

 daqp.separator_ = 0;

 daqp.counter_ = 0;
#endif
```

Constructor:

```
#ifdef DAQSOCKET1
 for(unsigned int i = 0; i < sdata_.size(); i++){
  if(sdata_[i].d_Name == "Message"){
   cerr <<  sdata_[i].d_Name.c_str() << ": ";
   for(unsigned int j = 0; j < MESSAGE_BUFFER - 1; j++)
    daqp.message_[j] = *((Char_t*)(sdata_[i].d_Data) + j);
   daqp.message_[MESSAGE_BUFFER - 1] = (Char_t)'\0';
   cerr << (char*) daqp.message_ << endl;
   IRenderer::getRenderer(i)->render((void*) daqp.message_, sdata_[i].d_Size);
  }else if(sdata_[i].d_Name == "Separator"){
   cerr <<  sdata_[i].d_Name.c_str() << ": ";
   daqp.separator_ = *(unsigned int*)(sdata_[i].d_Data);
   cerr <<  daqp.separator_ << endl;
   IRenderer::getRenderer(i)->render((void*) &daqp.separator_, sdata_[i].d_Size);

  }else if(sdata_[i].d_Name == "Counter"){
```

```
   cerr <<  sdata_[i].d_Name.c_str() << ": ";
   daqp.counter_ = *(unsigned int*)(sdata_[i].d_Data);
   cerr <<  daqp.counter_ << endl;
   IRenderer::getRenderer(i)->render((void*) &daqp.counter_, sdata_[i].d_Size);
  }
 }
#endif
```

Print data method:

```
#ifdef DAQSOCKET1
 cerr << "DAQSOCKET1 printData: " << endl;
 cerr << "Message: ";
 for(unsigned int i = 0; i < MESSAGE_BUFFER - 1; i++)
  cerr << *(((char*)daqp.message_) + i) << ", ";
 cerr << endl;

 cerr << "Separator: " << daqp.separator_ << endl;

 cerr << "Counter: " << daqp.counter_ << endl;
#endif
```

Chapter 6

EXAMPLE : acquisition of a stream from a socket (2)

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 SocketExample2.cpp) that after a simple handshake sends packets shaped like: <DAQ∗NNNNNNNN∗MMMMMMMM>, where <, > are respectively the start and the end syncronization character, and a couple of UINT32 random value (NNNNNNNN, MMMMMMMM) rendered respectively in a graph and in a histogram.

6.1 How does SocketExample2.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/SocketExample2 2000).

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

Checking the correct sintax for SocketExample2 execution: port number has to be passed as parameter

```
#ifdef DAQSOCKET2
 int sockfd, newsockfd, portno, clilen;
 char buffer[256];
 struct sockaddr_in serv_addr, cli_addr;
 int n;

 //Checking the port number
 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 2 random value, escaping the char and sending packets

```
while(1){
 unsigned int val1, val2;

 //srand(1);
 val1 = (unsigned int)((double)200 * rand() / RAND_MAX);

 //srand(2);
 val2 = (unsigned int)((double)200 * rand() / RAND_MAX);

 #ifdef DEBUG
 cerr << "First value: " << val1 << endl;
 cerr << "Second value: " << val2 << endl;
 #endif
```

```
unsigned int length_p = 0;
unsigned char packet[50];

packet[length_p++] = START;

packet[length_p++] = (unsigned char)'D';

packet[length_p++] = (unsigned char)'A';

packet[length_p++] = (unsigned char)'Q';

packet[length_p++] = (unsigned char)'*';

if(((unsigned char)((val1 >> 24) & 0xFF)) == START){
 packet[length_p++] = ESC;
 packet[length_p++] = START ^ ESC;
}else if(((unsigned char)((val1 >> 24) & 0xFF)) == END){
 packet[length_p++] = ESC;
 packet[length_p++] = packet[length_p] ^ ESC;
}else if(((unsigned char)((val1 >> 24) & 0xFF)) == ESC){
 packet[length_p++] = ESC;
 packet[length_p++] = ESC ^ ESC;
}else{
 packet[length_p++] = (unsigned char)((val1 >> 24) & 0xFF);
}

if(((unsigned char)((val1 >> 16) & 0xFF)) == START){
 packet[length_p++] = ESC;
 packet[length_p++] = START ^ ESC;
}else if(((unsigned char)((val1 >> 16) & 0xFF)) == END){
 packet[length_p++] = ESC;
 packet[length_p++] = END ^ ESC;
}else if(((unsigned char)((val1 >> 16) & 0xFF)) == ESC){
 packet[length_p++] = ESC;
 packet[length_p++] = ESC ^ ESC;
}else{
 packet[length_p++] = (unsigned char)((val1 >> 16) & 0xFF);
}


if(((unsigned char)((val1 >> 8) & 0xFF)) == START){
 packet[length_p++] = ESC;
 packet[length_p++] = START ^ ESC;
}else if(((unsigned char)((val1 >> 8) & 0xFF)) == END){
 packet[length_p++] = ESC;
 packet[length_p++] = END ^ ESC;
}else if(((unsigned char)((val1 >> 8) & 0xFF)) == ESC){
 packet[length_p++] = ESC;
 packet[length_p++] = ESC ^ ESC;
}else{
 packet[length_p++] = (unsigned char)((val1 >> 8) & 0xFF);
}

if(((unsigned char)((val1) & 0xFF)) == START){
 packet[length_p++] = ESC;
 packet[length_p++] = START ^ ESC;
}else if(((unsigned char)((val1) & 0xFF)) == END){
 packet[length_p++] = ESC;
 packet[length_p++] = END ^ ESC;
}else if(((unsigned char)((val1) & 0xFF)) == ESC){
 packet[length_p++] = ESC;
 packet[length_p++] = ESC ^ ESC;
}else{
 packet[length_p++] = (unsigned char)((val1) & 0xFF);
}

packet[length_p++] = (unsigned char)'*';
```

```
if(((unsigned char)((val2 >> 24) & 0xFF)) == START){
 packet[length_p++] = ESC;
 packet[length_p++] = START ^ ESC;
}else if(((unsigned char)((val2 >> 24) & 0xFF)) == END){
 packet[length_p++] = ESC;
 packet[length_p++] = packet[length_p] ^ ESC;
}else if(((unsigned char)((val2 >> 24) & 0xFF)) == ESC){
 packet[length_p++] = ESC;
 packet[length_p++] = ESC ^ ESC;
}else{
 packet[length_p++] = (unsigned char)((val2 >> 24) & 0xFF);
}

if(((unsigned char)((val2 >> 16) & 0xFF)) == START){
 packet[length_p++] = ESC;
 packet[length_p++] = START ^ ESC;
}else if(((unsigned char)((val2 >> 16) & 0xFF)) == END){
 packet[length_p++] = ESC;
 packet[length_p++] = packet[length_p] ^ ESC;
}else if(((unsigned char)((val2 >> 16) & 0xFF)) == ESC){
 packet[length_p++] = ESC;
 packet[length_p++] = ESC ^ ESC;
}else{
 packet[length_p++] = (unsigned char)((val2 >> 16) & 0xFF);
}

if(((unsigned char)((val2 >> 8) & 0xFF)) == START){
 packet[length_p++] = ESC;
 packet[length_p++] = START ^ ESC;
}else if(((unsigned char)((val2 >> 8) & 0xFF)) == END){
 packet[length_p++] = ESC;
 packet[length_p++] = packet[length_p] ^ ESC;
}else if(((unsigned char)((val2 >> 8) & 0xFF)) == ESC){
 packet[length_p++] = ESC;
 packet[length_p++] = ESC ^ ESC;
}else{
 packet[length_p++] = (unsigned char)((val2 >> 8) & 0xFF);
}

if((unsigned char)(val2 & 0xFF) == START){
 packet[length_p++] = ESC;
 packet[length_p++] = START ^ ESC;
}else if((unsigned char)(val2 & 0xFF) == END){
 packet[length_p++] = ESC;
 packet[length_p++] = packet[length_p] ^ ESC;
}else if((unsigned char)(val2 & 0xFF) == ESC){
 packet[length_p++] = ESC;
 packet[length_p++] = ESC ^ ESC;
}else{
 packet[length_p++] = (unsigned char)(val2 & 0xFF);
}

packet[length_p++] = END;

n = write(newsockfd, packet, length_p);
if (n < 0){
 cerr << "ERROR writing to socket" << endl;
 exit(1);
}else{
 cerr << "Packet size: " << dec << length_p << endl;
 cerr << "Packet sent: ";
 for(unsigned int k = 0; k < length_p; k++){
  cerr << hex << (int)packet[k] << ", ";
 }
 cerr << endl;
}
```

```
}

return 0;
```

6.2 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.

6.2.1 Stream title

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

6.2.2 Stream definition

In this case DAQ will acquire data from a socket:

- connected to the IP-address 'localhost' through the port number 2000;
- stream start character: 0x02;
- stream end character: 0x03;
- escape character: 0x20 (using xor decoder).

```
<stream stream_type="socket" stream_name="localhost" stream_info="2000">
 <stream_opt decoder="xor" start_char="02" end_char="03" esc_char="20" s_type_op
     t="hex"/>
</stream>
```

6.2.3 Handshake definition

The handshake type is 'start', the packet to send is:

- '1' (0x31)

ACK packet is:

- '1' (0x31)

and NAK packet is:

- '0' (0x30)

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

6.2.4 Payload definition

```
<packet_description length_pkt="13">
 <data data_name="Message" data_position="0" data_size="3" data_type="int8" type
     ="nothing" endianess="big-endian"/>
 <data data_name="Separator1" data_position="3" data_size="1" data_type="uint8"
     type="nothing" endianess="big-endian"/>
 <data data_name="Random1" data_position="4" data_size="4" data_type="uint32" ty
     pe="graph" endianess="big-endian">
  <graph_preset graph_name="Random1" graph_title="Random number graph" graph_max
     ="127"/>
  <canvas_preset canvas_title="Random1" canvas_width="400" canvas_height="400"/>

 </data>
 <data data_name="Separator2" data_position="8" data_size="1" data_type="uint8"
     type="nothing" endianess="big-endian"/>
 <data data_name="Random2" data_position="9" data_size="4" data_type="uint32" ty
     pe="histogram" endianess="big-endian">
  <histogram_preset histogram_name="Random2" histogram_title="Random number hist
     ogram" histogram_nbins="200" histogram_min="-0.5" histogram_max="200"/>
  <canvas_preset canvas_title="Random2" canvas_width="400" canvas_height="400"/>

 </data>
</packet_description>
```

6.2.5 socketLinux_2.xml

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

 <stream stream_type="socket" stream_name="localhost" stream_info="2000">
  <stream_opt decoder="xor" start_char="02" end_char="03" esc_char="20" s_type_op
      t="hex"/>
 </stream>

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

 <packet_description length_pkt="13">
  <data data_name="Message" data_position="0" data_size="3" data_type="int8" type
      ="nothing" endianess="big-endian"/>
  <data data_name="Separator1" data_position="3" data_size="1" data_type="uint8"
      type="nothing" endianess="big-endian"/>
  <data data_name="Random1" data_position="4" data_size="4" data_type="uint32" ty
      pe="graph" endianess="big-endian">
   <graph_preset graph_name="Random1" graph_title="Random number graph" graph_max
      ="127"/>
   <canvas_preset canvas_title="Random1" canvas_width="400" canvas_height="400"/>

  </data>
  <data data_name="Separator2" data_position="8" data_size="1" data_type="uint8"
      type="nothing" endianess="big-endian"/>
  <data data_name="Random2" data_position="9" data_size="4" data_type="uint32" ty
      pe="histogram" endianess="big-endian">
   <histogram_preset histogram_name="Random2" histogram_title="Random number hist
      ogram" histogram_nbins="200" histogram_min="-0.5" histogram_max="200"/>
   <canvas_preset canvas_title="Random2" canvas_width="400" canvas_height="400"/>

  </data>
 </packet_description>
```

```
</input_definition>
```

6.3 Payload analysis example

In this case the **DAQEvent** (p. 141) 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
```

6.4 Coding intervention

To configure DAQ is needed to modify the code in 2 files:

- **DAQPayload.h** (p. 234)
- **DAQSData.cxx** (p. 235)

6.4.1 Payload definition

Definition of buffer size:

```
#ifdef DAQSOCKET2
#define MESSAGE_BUFFER  4
#endif
```

Definition of the payload structure:

```
#ifdef DAQSOCKET2
 Char_t   message_[MESSAGE_BUFFER];    /**<Message arrived*/
 UChar_t   separator1_;   /**<Separator1 value*/
 UInt_t   random1_;   /**<Random1 value*/
 UChar_t   separator2_;   /**<Separator2 value*/
 UInt_t   random2_;   /**<Random2 value*/
#endif
```

6.4.2 DAQSData class (DAQSData.cxx)

Copy constructor:

```
#ifdef DAQSOCKET2
 for(int i = 0; i < MESSAGE_BUFFER; i++)
  daqp.message_[i] = orig.daqp.message_[i];

 daqp.separator1_ = orig.daqp.separator1_;

 daqp.random1_ = orig.daqp.random1_;

 daqp.separator2_ = orig.daqp.separator2_;

 daqp.random2_ = orig.daqp.random2_;
#endif
```

Default constructor:

```
#ifdef DAQSOCKET2
 for(int i = 0; i < MESSAGE_BUFFER; i++)
  daqp.message_[i] = 0;

 daqp.separator1_ = 0;

 daqp.random1_ = 0;

 daqp.separator2_ = 0;

 daqp.random2_ = 0;
#endif
```

Constructor:

```
#ifdef DAQSOCKET2
 for(unsigned int i = 0; i < sdata_.size(); i++){
  if(sdata_[i].d_Name == "Message"){
   cerr <<  sdata_[i].d_Name.c_str() << ": ";
   for(unsigned int j = 0; j < MESSAGE_BUFFER - 1; j++)
    daqp.message_[j] = *((Char_t*)(sdata_[i].d_Data) + j);
   daqp.message_[MESSAGE_BUFFER - 1] = (Char_t)'\0';
   cerr << (char*) daqp.message_ << endl;
   IRenderer::getRenderer(i)->render((void*) daqp.message_, sdata_[i].d_Size);
  }else if(sdata_[i].d_Name == "Separator1"){
   cerr <<  sdata_[i].d_Name.c_str() << ": ";
   daqp.separator1_ = *(unsigned int*)(sdata_[i].d_Data);
   cerr <<  daqp.separator1_ << endl;
   IRenderer::getRenderer(i)->render((void*) &daqp.separator1_, sdata_[i].d_Size)
      ;
  }else if(sdata_[i].d_Name == "Random1"){
   cerr <<  sdata_[i].d_Name.c_str() << ": ";
   daqp.random1_ = *(unsigned int*)(sdata_[i].d_Data);
   cerr <<  daqp.random1_ << endl;
   IRenderer::getRenderer(i)->render((void*) &daqp.random1_, sdata_[i].d_Size);
  }else if(sdata_[i].d_Name == "Separator2"){
   cerr <<  sdata_[i].d_Name.c_str() << ": ";
   daqp.separator2_ = *(unsigned int*)(sdata_[i].d_Data);
   cerr <<  daqp.separator2_ << endl;
   IRenderer::getRenderer(i)->render((void*) &daqp.separator2_, sdata_[i].d_Size)
      ;
  }else if(sdata_[i].d_Name == "Random2"){
   cerr <<  sdata_[i].d_Name.c_str() << ": ";
   daqp.random2_ = *(unsigned int*)(sdata_[i].d_Data);
   cerr <<  daqp.random2_ << endl;
   IRenderer::getRenderer(i)->render((void*) &daqp.random2_, sdata_[i].d_Size);
  }
 }
#endif
```

Print data method:

```
#ifdef DAQSOCKET2
 cerr << "DAQSOCKET2 printData: " << endl;
 cerr << "Message: ";
 for(unsigned int i = 0; i < MESSAGE_BUFFER - 1; i++)
  cerr << *(((char*)daqp.message_) + i) << ", ";
 cerr << endl;

 cerr << "Separator 1: " << daqp.separator1_ << endl;

 cerr << "Random 1: " << daqp.random1_ << endl;
```

```
 cerr << "Separator 2: " << daqp.separator2_ << endl;

 cerr << "Random 2: " << daqp.random2_ << endl;
#endif
```

Chapter 7

EXAMPLE : serial acquisition of a picture from HV7131GP and FLEX

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

7.1 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.

7.1.1 Stream title

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

7.1.2 Stream definition

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"/>
```

7.1.3 Handshake definition

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_typ
     e_pkt="hex"/>
 <handshake_pkt handshake_send="2A30312A30312A" handshake_ack="31" handshake_nac
     k="30" h_type_pkt="hex"/>
</handshake>
```

7.1.4 Payload definition

```
<packet_description length_pkt="19215">
 <data data_name="Header" data_position="0" data_size="3" data_type="uint8" type
     ="nothing" endianess="big-endian"/>
 <data data_name="Width" data_position="3" data_size="2" data_type="uint16" type
     ="nothing" endianess="big-endian"/>
 <data data_name="Height" data_position="5" data_size="2" data_type="uint16" typ
     e="nothing" endianess="big-endian"/>
 <data data_name="Duration" data_position="7" data_size="4" data_type="uint32" t
     ype="graph" endianess="big-endian">
  <graph_preset graph_name="Duration" graph_title="Frame duration (number of tic
     ks @40MIPS)" graph_max="20"/>
  <canvas_preset canvas_title="Duration" canvas_width="400" canvas_height="400"/
     >
 </data>
 <data data_name="Lost" data_position="11" data_size="2" data_type="uint16" type
     ="graph" endianess="big-endian">
  <graph_preset graph_name="Lost" graph_title="Frames lost during acquisition" g
     raph_max="20"/>
  <canvas_preset canvas_title="Lost" canvas_width="400" canvas_height="400"/>
 </data>
 <data data_name="YMean" data_position="13" data_size="2" data_type="uint16" typ
     e="graph" endianess="big-endian">
  <graph_preset graph_name="Luma" graph_title="Luminance average" graph_max="20"
     />
  <canvas_preset canvas_title="Luma" canvas_width="400" canvas_height="400"/>
 </data>
 <data data_name="Image" data_position="15" data_size="19200" data_type="uint8"
     type="image" endianess="big-endian">
  <image_preset image_name="Image" image_width="160" image_height="120" image_ch
     annel="1"/>
 </data>
</packet_description>
```

7.1.5 serialImageLinux.xml

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

 <stream stream_type="serial" stream_name="/dev/ttyUSB0" 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_typ
     e_pkt="hex"/>
  <handshake_pkt handshake_send="2A30312A30312A" handshake_ack="31" handshake_nac
     k="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" endianess="big-endian"/>
  <data data_name="Width" data_position="3" data_size="2" data_type="uint16" type
     ="nothing" endianess="big-endian"/>
  <data data_name="Height" data_position="5" data_size="2" data_type="uint16" typ
     e="nothing" endianess="big-endian"/>
  <data data_name="Duration" data_position="7" data_size="4" data_type="uint32" t
     ype="graph" endianess="big-endian">
   <graph_preset graph_name="Duration" graph_title="Frame duration (number of tic
     ks @40MIPS)" graph_max="20"/>
```

```
   <canvas_preset canvas_title="Duration" canvas_width="400" canvas_height="400"/
      >
  </data>
  <data data_name="Lost" data_position="11" data_size="2" data_type="uint16" type
      ="graph" endianess="big-endian">
   <graph_preset graph_name="Lost" graph_title="Frames lost during acquisition" g
      raph_max="20"/>
   <canvas_preset canvas_title="Lost" canvas_width="400" canvas_height="400"/>
  </data>
  <data data_name="YMean" data_position="13" data_size="2" data_type="uint16" typ
      e="graph" endianess="big-endian">
   <graph_preset graph_name="Luma" graph_title="Luminance average" graph_max="20"
      />
   <canvas_preset canvas_title="Luma" canvas_width="400" canvas_height="400"/>
  </data>
  <data data_name="Image" data_position="15" data_size="19200" data_type="uint8"
      type="image" endianess="big-endian">
   <image_preset image_name="Image" image_width="160" image_height="120" image_ch
      annel="1"/>
  </data>
 </packet_description>
</input_definition>
```

7.2 Payload analysis example

In this case the **DAQEvent** (p. 141) class is projected to store an image of 19200 bytes (160x120 pixel) and the other elements of the packet, 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
```

7.3 Coding intervention

To configure DAQ is needed to modify the code in 2 files:

- **DAQPayload.h** (p. 234)
- **DAQSData.cxx** (p. 235)

7.3.1 Payload definition

Definition of buffer size:

```
#ifdef DAQIMAGE
#define IMAGE_BUFFER  19200
#endif
```

Definition of the payload structure:

```
#ifdef DAQIMAGE
 UShort_t   width_;    /**<Image width*/
 UShort_t   height_;   /**<Image height*/
 ULong_t   duration_;   /**<Image duration*/
 UShort_t   lost_;    /**<Frame lost (1 true, 0 false)*/
 UShort_t   ymean_;    /**<Luminance mean value*/
```

```
 UChar_t   image_[IMAGE_BUFFER];  /**<Raw image*/
#endif
```

7.3.2 DAQSData class (DAQSData.cxx)

Copy constructor:

```
#ifdef DAQIMAGE
 daqp.width_ = orig.daqp.width_;
 daqp.height_ = orig.daqp.height_;
 daqp.duration_ = orig.daqp.duration_;
 daqp.lost_ = orig.daqp.lost_;
 daqp.ymean_ = orig.daqp.ymean_;
 memcpy ((UChar_t*)orig.daqp.image_, (UChar_t*)daqp.image_, IMAGE_BUFFER);
#endif
```

Default constructor:

```
#ifdef DAQIMAGE
 daqp.width_ = 0;
 daqp.height_ = 0;
 daqp.duration_ = 0;
 daqp.lost_ = 0;
 daqp.ymean_ = 0;
 memset(daqp.image_, '\0', IMAGE_BUFFER);
#endif
```

Constructor:

```
#ifdef DAQIMAGE
 for(unsigned int i = 0; i < sdata_.size(); i++){
  if(sdata_[i].d_Name == "Width"){
   cerr <<  sdata_[i].d_Name.c_str() << ": ";
   daqp.width_ = *((UShort_t*)(sdata_[i].d_Data));
   cerr <<  daqp.width_ << endl;
   IRenderer::getRenderer(i)->render((void*) &daqp.width_, sdata_[i].d_Size);
  }else if(sdata_[i].d_Name == "Height"){
   cerr <<  sdata_[i].d_Name.c_str() << ": ";
   daqp.height_ = *(UShort_t*)(sdata_[i].d_Data);
   cerr <<  daqp.height_ << endl;
   IRenderer::getRenderer(i)->render((void*) &daqp.height_, sdata_[i].d_Size);
  }else if(sdata_[i].d_Name == "Duration"){
   cerr <<  sdata_[i].d_Name.c_str() << ": ";
   daqp.duration_ = *(UInt_t*)(sdata_[i].d_Data);
   cerr <<  daqp.duration_ << endl;
   IRenderer::getRenderer(i)->render((void*) &daqp.duration_, sdata_[i].d_Size);
  }else if(sdata_[i].d_Name == "Lost"){
   cerr <<  sdata_[i].d_Name.c_str() << ": ";
   daqp.lost_ = *(UShort_t*)(sdata_[i].d_Data);
   cerr <<  daqp.lost_ << endl;
   IRenderer::getRenderer(i)->render((void*) &daqp.lost_, sdata_[i].d_Size);
  }else if(sdata_[i].d_Name == "YMean"){
   cerr <<  sdata_[i].d_Name.c_str() << ": ";
   daqp.ymean_ = *(UShort_t*)(sdata_[i].d_Data);
   cerr <<  daqp.ymean_ << endl;
   IRenderer::getRenderer(i)->render((void*) &daqp.ymean_, sdata_[i].d_Size);
  }else if(sdata_[i].d_Name == "Image"){
   cerr <<  sdata_[i].d_Name.c_str() << ": ";
   memcpy(daqp.image_, (UChar_t*)sdata_[i].d_Data, IMAGE_BUFFER);
   IRenderer::getRenderer(i)->render((void*) daqp.image_, sdata_[i].d_Size);
   cerr << endl;
```

```
  }
 }
#endif
```

Print data method:

```
#ifdef DAQIMAGE
 cerr << "DAQIMAGE printData: " << endl;
 cerr << "Width: " << daqp.width_ << endl;
 cerr << "Height: " << daqp.height_ << endl;
 cerr << "Duration: " << daqp.duration_ << endl;
 cerr << "Lost: " << daqp.lost_ << endl;
 cerr << "YMean: " << daqp.ymean_ << endl;
 cerr << "Image: " << endl;
 for (int k =0; k < IMAGE_BUFFER; k++){
  cerr << hex << (unsigned int)daqp.image_[k] << " ";
  if (k % 10 == 0) cerr << endl;
 }
 cerr << endl;
#endif
```

7.4 Post-processing images

The file LoopDAQEvent_Image.cpp is the application to process the images stored inside ROOT repository (default name is **test.root** (p. **244)**). The paradigm of this simple programm is:

- read the element

```
mypayload = mydaq->getPayload();
```

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

```
y[graph_abscissa] = fdy_cd(mypayload->image_, IMAGE_BUFFER);
TAxis *ax = hp->GetXaxis();
cerr << "Abscissa is: " << ax->GetBinCenter(graph_abscissa) << endl;
hp->SetBinContent((ax->GetBinCenter((Float_t)graph_abscissa)) + 2, y[graph_absc
    issa]);
hp->Draw("L");
canv_->Modified();
canv_->Update();
```

- show stored image

```
initRenderer(mypayload->width_, mypayload->height_);
renderImage((unsigned char*) mypayload->image_, mypayload->width_, mypayload->h
    eight_, 1);
```

Chapter 8

EXAMPLE : acquisition of a stream from a sniffer file

DAQ libraries give the possibilities read and parse data from a binary files, like the ones pruduced by the sniffer CC2420DK by Texas Instruments. In this case it is written an XML file to parse one of the files contained in the test_beacon folder: using the post-processing procedure is possible to display the beacon-coerency histogram for the OpenZB implementation over TinyOS.

8.1 XML file example for sniffer acquisition

This XML file permit to DAQ to read and parse a file generated by the CC2420DK sniffer.

8.1.1 Stream title

```
<definition title="Beacon interarrival time measurement with Open-ZB (over TinyO
    S)"/>
```

8.1.2 Stream definition

First of all is fixed the file-reading presets: the name of the file, the file structure... In this case DAQ will acquire data from the file:

- called test_beacon/BO1SO1.psd;
- the file header is long 4 bytes;
- the number of entries are written inside the header at position 0 (length 4 bytes).

```
<stream stream_type="file" stream_name="test_beacon/BO1SO1.psd">
 <file_stream header_l="4" n_entry_s="0" n_entry_l="4"/>
</stream>
```

8.1.3 Handshake definition

The handshake type is 'none'.

```
<handshake h_style="none"/>
```

8.1.4 Payload definition

```
<packet_description length_pkt="132">
 <data data_name="Timestamp" data_position="0" data_size="4" data_type="uint32"
     type="table" endianess="little-endian">
  <table_preset table_name="Timestamp"/>
 </data>
 <data data_name="Payload" data_position="4" data_size="128" data_type="uint8" t
     ype="nothing" endianess="little-endian"/>
</packet_description>
```

8.1.5 XML file example for sniffer acquisition

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<input_definition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNam
      espaceSchemaLocation="input.xsd">
 <definition title="Beacon interarrival time measurement with Open-ZB (over TinyO
      S)"/>

 <stream stream_type="file" stream_name="test_beacon/BO1SO1.psd">
  <file_stream header_l="4" n_entry_s="0" n_entry_l="4"/>
 </stream>
 <handshake h_style="none"/>

 <packet_description length_pkt="132">
  <data data_name="Timestamp" data_position="0" data_size="4" data_type="uint32"
      type="table" endianess="little-endian">
   <table_preset table_name="Timestamp"/>
  </data>
  <data data_name="Payload" data_position="4" data_size="128" data_type="uint8" t
      ype="nothing" endianess="little-endian"/>
 </packet_description>
</input_definition>
```

8.2 Payload analysis example

In this case the **DAQEvent** (p. 141) 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
```

8.3 Coding intervention

To configure DAQ is needed to modify the code in 2 files:

- **DAQPayload.h** (p. 234)
- **DAQSData.cxx** (p. 235)

8.3.1 Payload definition

Definition of buffer size:

```
#ifdef DAQSNIFFER
#define PAYLOAD_BUFFER  128
#endif
```

Definition of the payload structure:

```
#ifdef DAQSNIFFER
 UInt_t   timestamp_;   /**<Beacon timestamp*/
 UChar_t   payload_[PAYLOAD_BUFFER]; /**<Packet payload*/
#endif
```

8.3.2 DAQSData class (DAQSData.cxx)

Copy constructor:

```
#ifdef DAQSNIFFER
 daqp.timestamp_=orig.daqp.timestamp_;
 memcpy ((UChar_t*)orig.daqp.payload_, (UChar_t*)daqp.payload_, PAYLOAD_BUFFER);
#endif
```

Default constructor:

```
#ifdef DAQSNIFFER
 daqp.timestamp_ = 0;
 memset(daqp.payload_, '\0', PAYLOAD_BUFFER);
#endif
```

Constructor:

```
#ifdef DAQSNIFFER
 for(unsigned int i = 0; i < sdata_.size(); i++){
  if(sdata_[i].d_Name == "Timestamp"){
   cerr <<  sdata_[i].d_Name.c_str() << ": ";
   daqp.timestamp_ = *(UInt_t*)(sdata_[i].d_Data);
   cerr << dec <<  daqp.timestamp_ << endl;
   IRenderer::getRenderer(i)->render((void*) &daqp.timestamp_, sdata_[i].d_Size);

  }else if(sdata_[i].d_Name == "Payload"){
   cerr <<  sdata_[i].d_Name.c_str() << ": ";
   memcpy(daqp.payload_, (UChar_t*)sdata_[i].d_Data, PAYLOAD_BUFFER);
   IRenderer::getRenderer(i)->render((void*)daqp.payload_, sdata_[i].d_Size);
   /*for (int k =0; k < PAYLOAD_BUFFER; k++){
    cerr << hex << (unsigned int)daqp.payload_[k] << " ";
    if (k % 10 == 0) cerr << endl;
   }*/
   cerr << endl;
  }
 }
#endif
```

Print data method:

```
#ifdef DAQSNIFFER
 cerr << "DAQSNIFFER printData: " << endl;
 cerr << "Time-stamp: : " << daqp.timestamp_ << endl;
 cerr << "Payload: " << endl;
 for (int k =0; k < PAYLOAD_BUFFER; k++){
  cerr << hex << (unsigned int)daqp.payload_[k] << " ";
  if (k % 10 == 0) cerr << endl;
 }
 cerr << endl;
#endif
```

Chapter 9

Todo List

Class CFile (p. **63**) Implement trailer handling

Member CFile::getDefinition (p. **65**)**(struct SDef** (p. **171**) **def, s_read_style sty)**
endianess in header reading

Member CFile::retStream (p. **65**)**(unsigned char** ∗∗**buffer, unsigned long** ∗**length)**
Introduce other feof check and ferror controls

Class CGraph (p. **75**) Add possible fitting and error handler

Add some style stuff

Class CHistogram (p. **81**) Add possible fitting and error handler

Add some style stuff

Member CHistogram::fillHisto (p. **84**)**(double data)** Insert an entire buffer like image histogram

Class CImage (p. **87**) Add possible compressions and formats

Add Targets (one or more)

Data type switching

Member CReader::getDefinition (p. **103**)**(struct SDef** (p. **171**) **def)** usb

stdin

Member initRenderer (p. **264**) Add parameter to dimension the size of the windows

Member initRenderer_black (p. **265**) Add parameter to dimension the size of the windows

Chapter 10

Class Index

10.1 Class Hierarchy

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

CDaq . . . 59
CXMLParser . . . 133
DAQPayload . . . 145
ErrorHandler . . . 151
  DOMTreeErrorReporter . . . 148
IRenderer::Factory . . . 152
IDecoderDaq . . . 154
  CXor . . . 138
IFormatter . . . 156
  CFormatter . . . 68
IMessageDaq . . . 158
  CMessager . . . 92
IRenderer . . . 160
  CGraph . . . 75
  CHistogram . . . 81
  CImage . . . 87
  CNone . . . 94
  CTable . . . 117
  CTarget . . . 121
IStreamDaq . . . 163
  CFile . . . 63
  CPipe . . . 97
  CSerial . . . 105
  CSocket . . . 111
IStreamReader . . . 165
  CReader . . . 102
IXml . . . 167
  CXml . . . 125
SDef . . . 171
SFile . . . 174
SHandshaking . . . 176

SMData 178
SMessage 180
SOption 181
Srs_232 183
StrX 185
TObject 187
DAQEvent 141
DAQSData 146
SData 169

Chapter 11

Class Index

11.1 Class List

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

CDaq (User interface from the data acquisition library and the user application) . . . 59
CFile (Implementation of IStream interface to read a file. This class reads a file and creates packet shaped like the **SDef** (p. 171) structure directives) 63
CFormatter (Class to format data received from stream based on the metadata vector (created by the xml parser)) 68
CGraph (Implementation of **IRenderer** (p. 160) interface to render an graph using root libraries) 75
CHistogram (Implementation of **IRenderer** (p. 160) interface to render an histogram using root libraries) 81
CImage (Implementation of **IRenderer** (p. 160) interface to render a raw image using FLTK libraries) 87
CMessager (Implementation of **IMessageDaq** (p. 158) interface to devide in messages a packet) 92
CNone (Implementation of **IRenderer** (p. 160) interface to handle data with no semantic. This class is implemented to discrard not important character in the stream (like separator)) 94
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. 171) structure directives. Daq is client of the pipe created by the server thread) 97
CReader (Implemenation for **IStreamReader** (p. 165) interface to read, decode and divide packet to be passed to formatter. This class uses instantiation of the classes **IStreamDaq** (p. 163), **IDecoderDaq** (p. 154), **IMessageDaq** (p. 158)) 102
CSerial (Implementation of IStream interface to read from serial (RS232). This class reads a file and creates packet shaped like the **SDef** (p. 171) structure directives) 105
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. 171) structure directives) 111
CTable (Implementation of **IRenderer** (p. 160) interface to write a table inside a file) 117
CTarget (Implementation of **IRenderer** (p. 160) interface to render a black screen to show the position of a target using FLTK libraries) 121
CXml (DAQ specific xml parsing class: class for xml parsing using xml_schema input.xsd) 125

CXMLParser (Class to implement methods to init, parse, process an xml file. Genercic class) . . . 133
CXor (Implementation of **IDecoderDaq** (p. 154) interface to decode the escape character with xor function) . . . 138
DAQEvent (Class containing members and methods for a complete event acquired through the link) . . . 141
DAQPayload (Structure that describe the packet payload according with xml file) . . . 145
DAQSData (Class containing members and methods for the data acquired through the link) . . . 146
DOMTreeErrorReporter (This class registers as an **ErrorHandler** (p. 151) with the DOM parser and reports errors to the application) . . . 148
ErrorHandler (Basic interface for SAX error handlers. From the Xerces package) . . . 151
IRenderer::Factory (**IRenderer** (p. 160) object factory to handle the renderer) . . . 152
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. 102) class) . . . 154
IFormatter (Interface for data formatter) . . . 156
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. 102) class) . . . 158
IRenderer (Interface to render data (images, histograms, graphs, tables)) . . . 160
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. 102) class) . . . 163
IStreamReader (Interface for different type of stream reader) . . . 165
IXml (Interface for xml parser) . . . 167
SData (Structure of data elements) . . . 169
SDef (Structure for stream definition) . . . 171
SFile (Structure of File) . . . 174
SHandshaking (Structure of handshaking packets) . . . 176
SMData (Structure of metadata elements) . . . 178
SMessage (Structure of message) . . . 180
SOption (Structure of steam option: syncro character) . . . 181
Srs_232 (Structure of RS-232 Preset) . . . 183
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) . . . 185
TObject (Base class in the ROOT framework for every serializable objects) . . . 187

Chapter 12

File Index

12.1 File List

Here is a list of all files with brief descriptions:

CDaq.cpp (**CDaq** (p. 59) methods development) 189
CDaq.h (User interface from the data acquisition library and the user application) . . 191
CFile.cpp (File reading functions development) 192
CFile.h (Implementation of IStream interface to read a file. This class reads a file and creates packet shaped like the **SDef** (p. 171) structure directives) 193
CFormatter.cpp (Formatting functions development) 194
CFormatter.h (Class to format data received from stream based on the metadata vector (created by the xml parser)) 195
CGraph.cpp (Graph class method development) 196
CGraph.h (Class to render an graph using the root libraries) 197
CHistogram.cpp (Histogram class method development) 198
CHistogram.h (Class to render a an histogram using the root libraries) 199
CImage.cpp (Image rendering class method development) 200
CImage.h (Class to render a raw image using FLTK libraries) 201
CMessager.cpp (Divisor in messages functions development) 202
CMessager.h (Implementation of **IMessageDaq** (p. 158) interface to devide in messages a packet) 203
CNone.cpp (Registering "none renderer" in the object factory) 204
CNone.h (Class to handle data with no semantic) 205
CPipe.cpp (Pipe access functions develpoment) 206
CPipe.h (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. 171) structure directives. Daq is client of the pipe created by the server thread) 207
CReader.cpp (Reader functions development) 208
CReader.h (Implemenation for **IStreamReader** (p. 165) interface to read, decode and divide packet to be passed to formatter. This class uses instantiation of the classes **IStreamDaq** (p. 163), **IDecoderDaq** (p. 154), **IMessageDaq** (p. 158)) 209
CSerial.cpp (Serial access functions development) 210
CSerial.h (Implementation of IStream interface to read from serial (RS232). This class reads a file and creates packet shaped like the **SDef** (p. 171) structure directives)211
CSocket.cpp (Socket access functions develpoment) 212

CSocket.h (Implementation of IStream interface to read a file. This class reads a file and creates packet shaped like the **SDef** (p. 171) structure directives) 213
CTable.cpp (**CTable** (p. 117) class method development) 214
CTable.h (Class to write a table inside a file) 215
CTarget.cpp (Target rendering class method development) 216
CTarget.h (Class to render a black screen to show the position of a target) 217
CXml.cpp (Xml parser) 218
CXml.h (DAQ specific xml parsing class: class for xml parsing using xml_schema input.xsd) 219
CXmlParser.cpp (XML parsing and analysis development functions) 220
CXmlParser.h (Class to implement methods to init, parse, process an xml file. Genercic class) 221
CXor.cpp (**CXor** (p. 138) decoder development functions) 222
CXor.h (Implementation of **IDecoderDaq** (p. 154) interface to decode the escape character with xor function) 223
DAQ_enum.h (Enumerations for the UI **CDaq** (p. 59) and data types definition) . . 224
DAQ_struct.h (General structures used by DAQ) 227
DAQError.h (Error enumerations for the UI **CDaq** (p. 59)) 228
DAQEvent.cxx (**DAQEvent** (p. 141) functions development) 231
DAQEvent.h (This file contains the class to acquire an event through the link) 232
DAQEventLinkDef.h (Compiling directive to generate the dictionary) 233
DAQPayload.h (This file contains the stucture to store the data according with xml file)234
DAQSData.cxx (**DAQSData** (p. 146) functions development) 235
DAQSData.h (This file contains information on the DAQ classes) 236
Data_struct.h (Structure of formatted data: data formatted are stored inside **SData** (p. 169) vectors) 237
documentation.h (Documentation pages) 238
DOMTreeErrorReporter.cpp (**DOMTreeErrorReporter** (p. 148) functions development) 239
DOMTreeErrorReporter.hpp (Development of the classes **DOMTreeErrorReporter** (p. 148) and **StrX** (p. 185)) 240
fdy.cpp (Vision algorithms development functions) 241
fdy.h (This function calculates the percentage of similarity using FDY algorithm) . . . 242
form_enum.h (Header to enumerate the return value of **CFormatter** (p. 68) class) . 243
globals.cxx (Globals functions & variables) 244
globals.h (Globals functions & variables) 245
IDecoderDaq.h (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. 102) class) 246
IFormatter.h (Interface for data formatter) 247
IMessageDaq.h (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. 102) class) 248
IRenderer.cpp (**IRenderer** (p. 160) class not-virtual method development) 249
IRenderer.h (Interface to render data (images, histograms, graphs, tables)) 250
IStreamDaq.h (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. 102) class) 251
IStreamReader.h (Interface for different type of stream reader) 252
IXml.h (Interface for xml parser and analyzer) 253
myUtils.cpp (Vision algorithms development functions) 254
myUtils.h (Conversion functions. This is library usefull for image compression and image conversion) 257

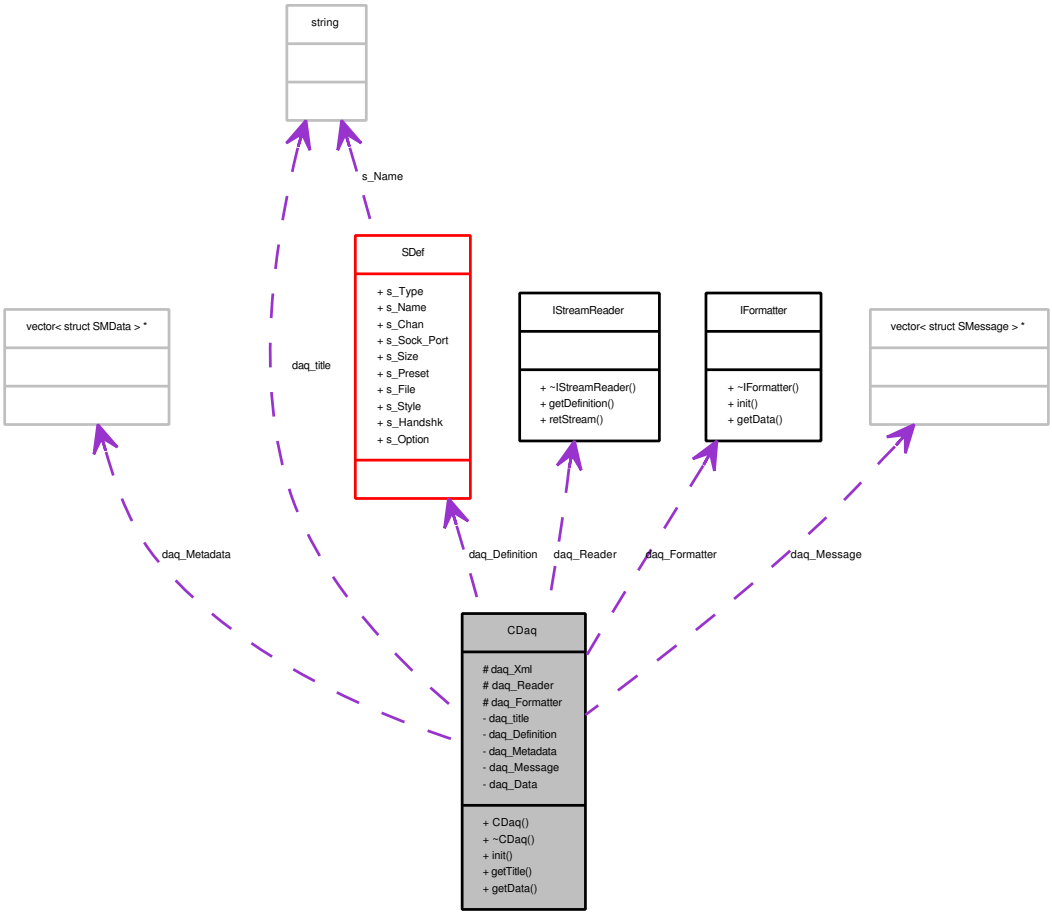
ROOTInitializer.cpp (Static class to initialize the repository file) 261
Show_image.cpp (Image rendering function development) 262
Show_image.h (Function to render a raw image using FLTK libraries) 264
Stream_enum.h (All enumeration used by the data reader) 267
XML_enum.h (Enumeration for Xml file parsing and analysis) 269

Chapter 13

Class Documentation

13.1 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**

13.1.1 Constructor & Destructor Documentation

13.1.1.1 CDaq::CDaq (char * *xmlFile*)

CDaq (p. 59) constructor.

Parameters:

xmlFile pointer to the string of the xml file path

13.1.1.2 CDaq::~CDaq ()

CDaq (p. 59) destructor.

13.1.2 Member Function Documentation

13.1.2.1 daq_ret CDaq::init (void)

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

13.1.2.2 string CDaq::getTitle (void)

Method that returns the title of the title.

Returns:

Returns the string of the title

13.1.2.3 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

13.1.3 Member Data Documentation

13.1.3.1 IXml* CDaq::daq_Xml [protected]

Instance of the xml parser interface.

13.1.3.2 IStreamReader* CDaq::daq_Reader [protected]

Instance of data reader interface.

13.1.3.3 IFormatter* CDaq::daq_Formatter [protected]

Instance of the formatter interface

13.1.3.4 string CDaq::daq_title [private]

Variable that contains the title

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

Varibale of **SDef** (p. 171) struct that define the stream

See also:

SDef (p. 171)

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

Vector of metadata

See also:

SMData (p. 178)

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

Vector of the messages (output of the reader)

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

Formatted data vector

See also:

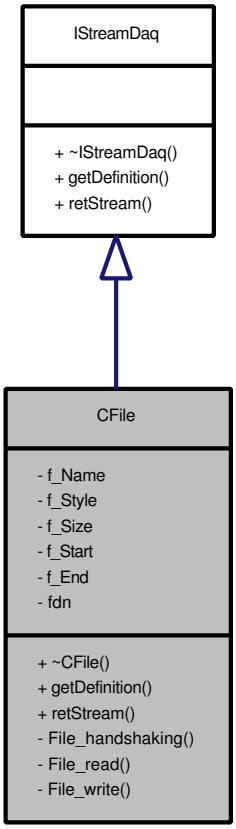
SData (p. 169)

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

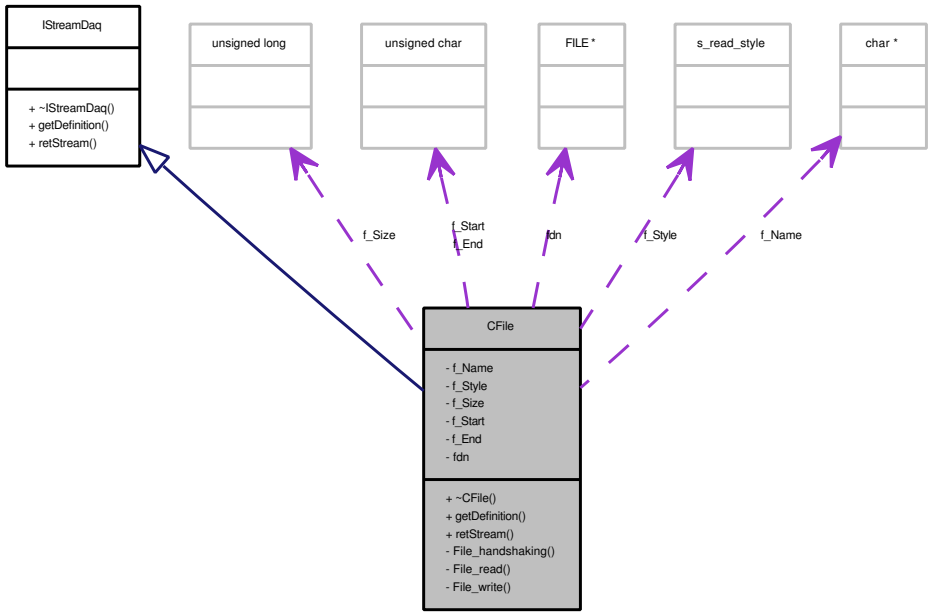
- **CDaq.h**
- **CDaq.cpp**

13.2 CFile Class Reference

Implementation of IStream interface to read a file. This class reads a file and creates packet shaped like the **SDef** (p. 171) 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)
- **s_ret File_read** (unsigned char *data, unsigned int data_size)
- **s_ret File_write** (unsigned char *data, unsigned int data_size)

Private Attributes

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

13.2.1 Detailed Description

Todo

Implement trailer handling

13.2.2 Constructor & Destructor Documentation

13.2.2.1 CFile::~CFile ()

CFile (p. 63) destructor.

13.2.3 Member Function Documentation

13.2.3.1 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. 267)

Todo

endianess in header reading

Implements **IStreamDaq** (p. 163).

13.2.3.2 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. 267)

Todo

Introduce other feof check and ferror controls

Implements **IStreamDaq** (p. 164).

13.2.3.3 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. 267)

13.2.3.4 s_ret CFile::File_read (unsigned char * *data*, unsigned int *data_size*) [private]

This function read a stream from a file.

Parameters:

data buffer to store data

data_size size in bytes

Returns:

s_ret value about sending from

See also:

s_ret (p. 267)

13.2.3.5 s_ret CFile::File_write (unsigned char * *data*, unsigned int *data_size*) [private]

This function write a stream in a file.

Parameters:

data buffer of data data

data_size size in bytes

Returns:

s_ret value about sending from

See also:

s_ret (p. 267)

13.2.4 Member Data Documentation

13.2.4.1 char* CFile::f_Name [private]

File path

13.2.4.2 s_read_style CFile::f_Style [private]

Reading style

13.2.4.3 unsigned long CFile::f_Size [private]

Packet total size

13.2.4.4 unsigned char CFile::f_Start [private]

Start character

13.2.4.5 unsigned char CFile::f_End [private]

End character

13.2.4.6 FILE* CFile::fdn [private]

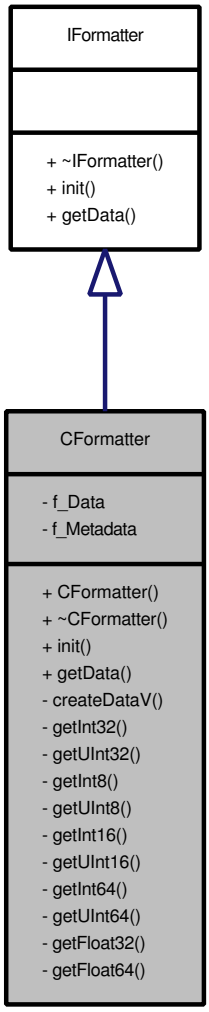
File descriptor number

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

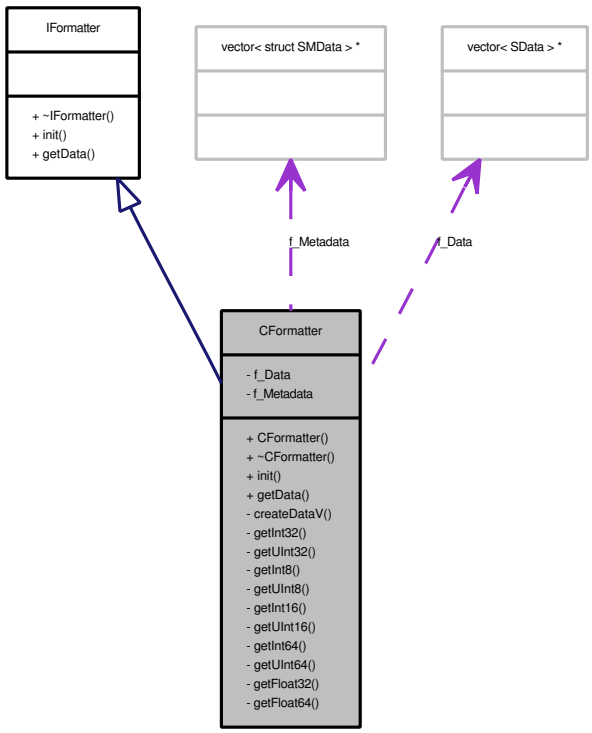
- **CFile.h**
- **CFile.cpp**

13.3 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** (std::vector< **SData** > *data)
- virtual **f_ret init** (std::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, **endianess_type** _endianess)
- unsigned int * **getUInt32** (unsigned char *v, unsigned long _size, **endianess_type** _endianess)
- char * **getInt8** (unsigned char *v, unsigned long _size, **endianess_type** _endianess)
- unsigned char * **getUInt8** (unsigned char *v, unsigned long _size, **endianess_type** _endianess)
- short * **getInt16** (unsigned char *v, unsigned long _size, **endianess_type** _endianess)
- unsigned short * **getUInt16** (unsigned char *v, unsigned long _size, **endianess_type** _endianess)
- long * **getInt64** (unsigned char *v, unsigned long _size, **endianess_type** _endianess)
- unsigned long * **getUInt64** (unsigned char *v, unsigned long _size, **endianess_type** _endianess)
- float * **getFloat32** (unsigned char *v, unsigned long _size, **endianess_type** _endianess)
- double * **getFloat64** (unsigned char *v, unsigned long _size, **endianess_type** _endianess)

Private Attributes

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

13.3.1 Detailed Description

See also:

SMData (p. 178)

13.3.2 Constructor & Destructor Documentation

13.3.2.1 CFormatter::CFormatter (std::vector< SData > * *data*)

Constructor.

Parameters:

header Pointer to header string vector

data Pointer to formatted data vector

13.3.2.2 CFormatter::~CFormatter ()

Destructor.

13.3.3 Member Function Documentation

13.3.3.1 f_ret CFormatter::init (std::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. 243)

Implements **IFormatter** (p. 157).

13.3.3.2 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. 165) (input)

Returns:

One of the possible value of f_ret

See also:

f_ret (p. 243)

Implements **IFormatter** (p. 157).

13.3.3.3 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. 243)

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

Method to format integer values.

Parameters:

v Pointer to a subset of stream data (input)

_size Size of the resulting integer vector (input)

_endianess Endianess type

Returns:

This method returns the pointer of the integer vector

13.3.3.5 unsigned int * CFormatter::getUInt32 (unsigned char * *v*, unsigned long *_size*, endianess_type *_endianess*) [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)

_endianess Endianess type (input)

Returns:

This method returns the pointer of the unsigned integer vector

13.3.3.6 char * CFormatter::getInt8 (unsigned char * *v*, unsigned long *_size*, endianess_type *_endianess*) [private]

Method to format char values.

Parameters:

v Pointer to a subset of stream data (input)

_size Size of the resulting char vector (input)

_endianess Endianess type (input)

Returns:

This method returns the pointer of the char vector

13.3.3.7 unsigned char * CFormatter::getUInt8 (unsigned char * *v*, unsigned long *_size*, endianess_type *_endianess*) [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)

_endianess Endianess type (input)

Returns:

This method returns the pointer of the unsigned char vector

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

Method to format short values.

Parameters:

v Pointer to a subset of stream data (input)

_size Size of the resulting short vector (input)

_endianess Endianess type (input)

Returns:

This method returns the pointer of the char vector

13.3.3.9 unsigned short * CFormatter::getUInt16 (unsigned char * *v*, unsigned long *_size*, endianess_type *_endianess*) [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)

_endianess Endianess type (input)

Returns:

This method returns the pointer of the unsigned char vector

13.3.3.10 long * CFormatter::getInt64 (unsigned char * *v*, unsigned long *_size*, endianess_type *_endianess*) [private]

Method to format long values.

Parameters:

v Pointer to a subset of stream data (input)

_size Size of the resulting long vector (input)

_endianess Endianess type (input)

Returns:

This method returns the pointer of the long vector

13.3.3.11 unsigned long * CFormatter::getUInt64 (unsigned char * *v*, unsigned long *_size*, endianess_type *_endianess*) [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)

_endianess Endianess type (input)

Returns:

This method returns the pointer of the unsigned long vector

13.3.3.12 float * CFormatter::getFloat32 (unsigned char * *v*, unsigned long *_size*, endianess_type *_endianess*) [private]

Method to format float values.

Parameters:

v Pointer to a subset of stream data (input)

_size Size of the resulting float vector (input)

_endianess Endianess type (input)

Returns:

This method returns the pointer of the float vector

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

Method to format double values.

Parameters:

v Pointer to a subset of stream data (input)

_size Size of the resulting double vector (input)

_endianess Endianess type (input)

Returns:

This method returns the pointer of the double

13.3.4 Member Data Documentation

13.3.4.1 std::vector<SData>* CFormatter::f_Data [private]

SData (p. 169) vector

See also:

SData (p. 169)

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

Metadata vector

See also:

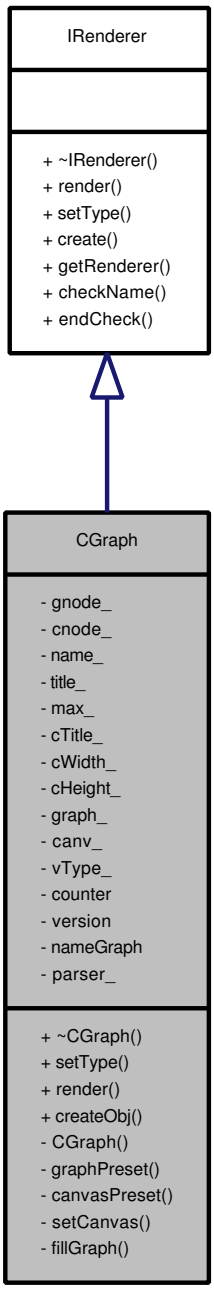
SMData (p. 178)

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

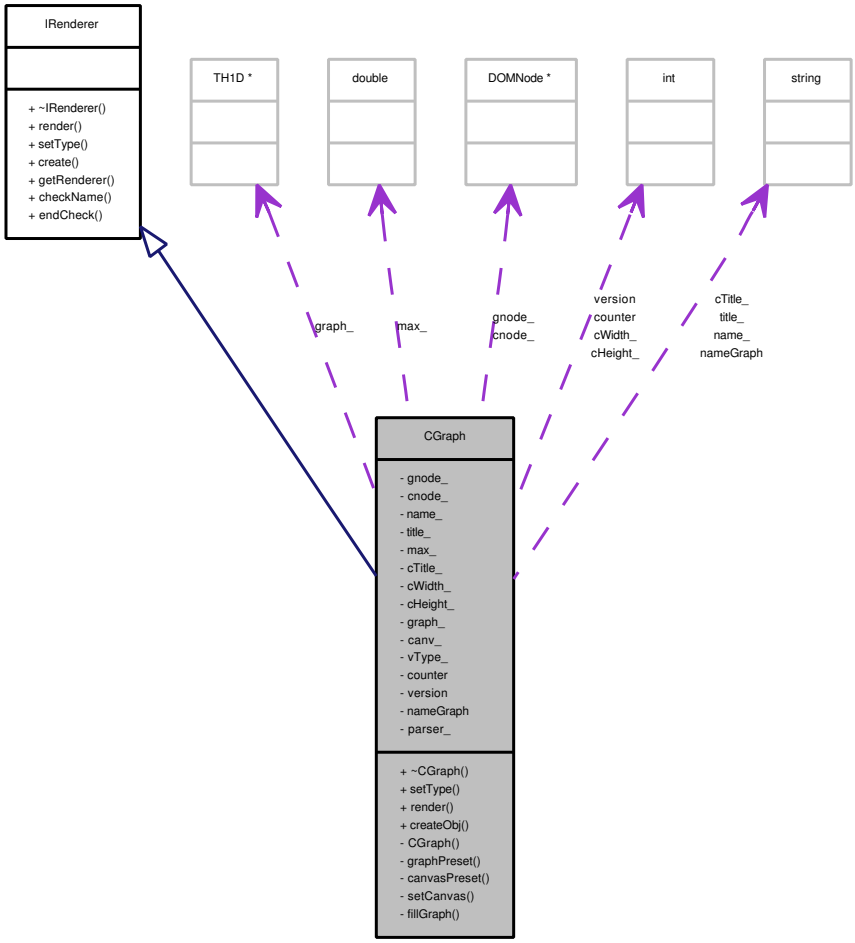
- **CFormatter.h**
- **CFormatter.cpp**

13.4 CGraph Class Reference

Implementation of **IRenderer** (p. 160) interface to render an graph using root libraries. Inheritance diagram for CGraph:



Collaboration diagram for CGraph:



Public Member Functions

- void **setType** (**var_type** vType)
- void **render** (void *data, unsigned long size)

Static Public Member Functions

- static **IRenderer** * **createObj** (const void *node)

Private Member Functions

- **CGraph** (const DOMNode *node)
- void **graphPreset** (const DOMNode *node)
- void **canvasPreset** (const DOMNode *node)
- void **fillGraph** (double data)

Private Attributes

- DOMNode * **gnode_**
- DOMNode * **cnode_**
- std::string **name_**
- std::string **title_**
- double **max_**
- std::string **cTitle_**
- unsigned int **cWidth_**
- unsigned int **cHeight_**
- TH1D * **graph_**
- TCanvas * **canv_**
- **var_type vType_**
- unsigned int **counter**
- unsigned int **version**
- std::string **nameGraph**
- **CXMLParser** * **parser_**

13.4.1 Detailed Description

Todo

Add possible fitting and error handler

Todo

Add some style stuff

13.4.2 Constructor & Destructor Documentation

13.4.2.1 CGraph::~CGraph (void)

13.4.2.2 CGraph::CGraph (const DOMNode * *node*) [private]

The constructor function parses the xml node relative to graph and canvas and intializes those.

Parameters:

node pointer to parent DOM node

13.4.3 Member Function Documentation

13.4.3.1 IRenderer * CGraph::createObj (const void * *node*) [static]

This function, used by the object factory, return an istance of **CGraph** (p. 75) class.

Parameters:

node pointer to a generic node (in this case DOMNode) where extracts the graph presets.

13.4.3.2 void CGraph::setType (var_type *vType*) [virtual]

Function to set the data type.

Parameters:

data pointer to the vector/scalar to be mapped on histogram

Implements **IRenderer** (p. 161).

13.4.3.3 void CGraph::render (void * *data*, unsigned long *size*) [virtual]

Function to render the histogram.

Parameters:

data pointer to the vector/scalar to be mapped on histogram

Implements **IRenderer** (p. 161).

13.4.3.4 void CGraph::graphPreset (const DOMNode * *node*) [private]

This function parse the xml-file to extract the graph presets.

Parameters:

node pointer to parent DOM node

13.4.3.5 void CGraph::canvasPreset (const DOMNode * *node*) [private]

This function parse the xml-file to extract the canvas presets.

Parameters:

node pointer to parent DOM node

13.4.3.6 void CGraph::setCanvas (void) [private]

Construct the root canvas.

13.4.3.7 void CGraph::fillGraph (double *data*) [private]

Function to fill the histogram.

Parameters:

data pointer to the vector/scalar to be mapped on histogram

13.4.4 Member Data Documentation

13.4.4.1 DOMNode* CGraph::gnode_ [private]

Graph presets DOM node

13.4.4.2 DOMNode* CGraph::cnode_ [private]

Canvas presets DOM node

13.4.4.3 std::string CGraph::name_ [private]

Graph name

13.4.4.4 std::string CGraph::title_ [private]

Graph title

13.4.4.5 double CGraph::max_ [private]

x max value

13.4.4.6 std::string CGraph::cTitle_ [private]

Canvas title

13.4.4.7 unsigned int CGraph::cWidth_ [private]

Canvas width

13.4.4.8 unsigned int CGraph::cHeight_ [private]

Canvas height

13.4.4.9 TH1D* CGraph::graph_ [private]

Pointer to a ROOT histogram (Double type)

13.4.4.10 TCanvas* CGraph::canv_ [private]

Pointer to a ROOT canvas

13.4.4.11 var_type CGraph::vType_ [private]

Graph variable type

13.4.4.12 unsigned int CGraph::counter [private]

Counter for time

13.4.4.13 unsigned int CGraph::version [private]

Graph version

13.4.4.14 std::string CGraph::nameGraph [private]

Graph name

13.4.4.15 CXMLParser* CGraph::parser_ [private]

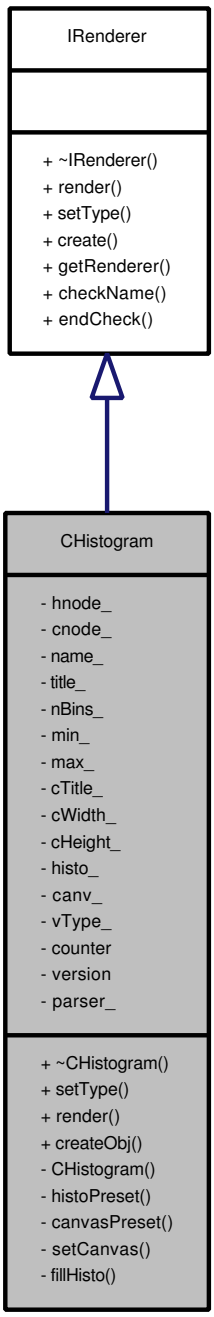
Pointer to Xml parser

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

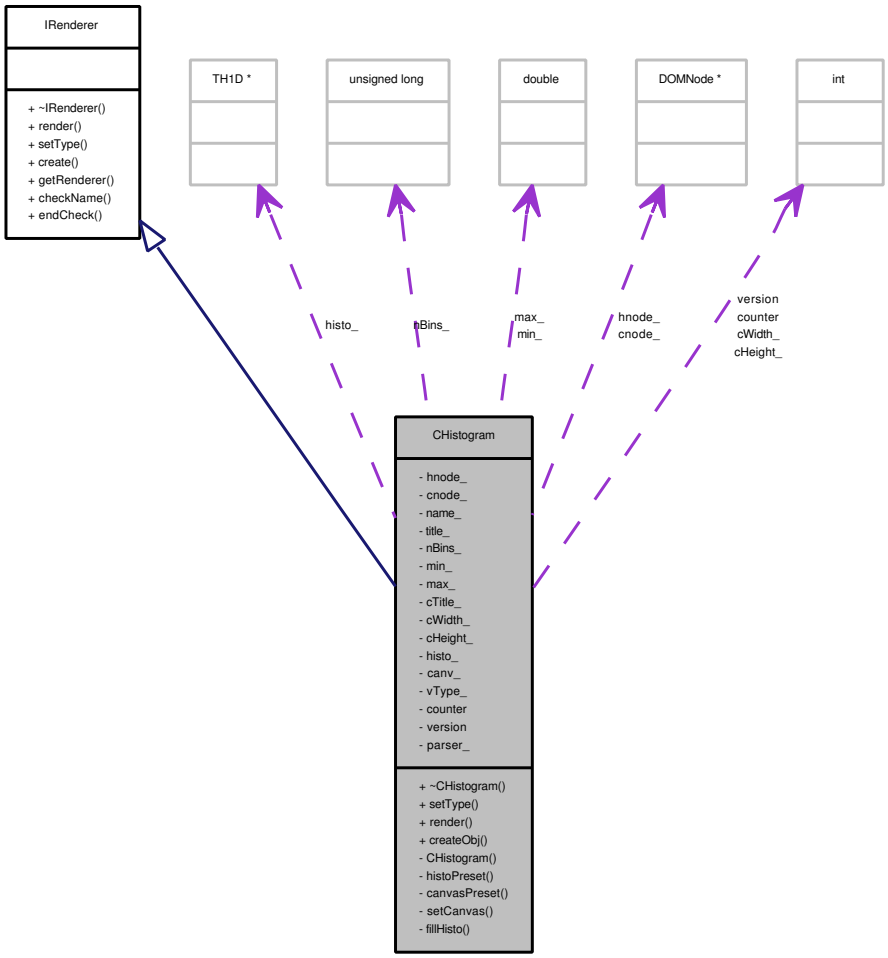
- **CGraph.h**
- **CGraph.cpp**

13.5 CHistogram Class Reference

Implementation of **IRenderer** (p. 160) interface to render an histogram using root libraries. Inheritance diagram for CHistogram:



Collaboration diagram for CHistogram:



Public Member Functions

- void **setType** (**var_type** vType)
- void **render** (void *data, unsigned long size)

Static Public Member Functions

- static **IRenderer** * **createObj** (const void *node)

Private Member Functions

- **CHistogram** (const DOMNode *node)
- void **histoPreset** (const DOMNode *node)
- void **canvasPreset** (const DOMNode *node)
- void **fillHisto** (double data)

Private Attributes

- DOMNode * **hnode_**
- DOMNode * **cnode_**
- std::string **name_**
- std::string **title_**
- unsigned long **nBins_**
- double **min_**
- double **max_**
- std::string **cTitle_**
- unsigned int **cWidth_**
- unsigned int **cHeight_**
- TH1D * **histo_**
- TCanvas * **canv_**
- **var_type vType_**
- unsigned int **counter**
- unsigned int **version**
- **CXMLParser * parser_**

13.5.1 Detailed Description

Todo

Add possible fitting and error handler

Todo

Add some style stuff

13.5.2 Constructor & Destructor Documentation

13.5.2.1 CHistogram::~CHistogram (void)

13.5.2.2 CHistogram::CHistogram (const DOMNode * *node*) [private]

The constructor function parses the xml node relative to histogram and canvas and intializes those.

Parameters:

node pointer to parent DOM node

13.5.3 Member Function Documentation

13.5.3.1 IRenderer * CHistogram::createObj (const void * *node*) [static]

This function, used by the object factory, return an istance of **CHistogram** (p. 81) class.

Parameters:

node pointer to a generic node (in this case DOMNode) where extracts the histogram presets.

13.5.3.2 void CHistogram::setType (var_type *vType*) [virtual]

Function to set the data type.

Parameters:

vType data type

Implements **IRenderer** (p. 161).

13.5.3.3 void CHistogram::render (void * *data*, unsigned long *size*) [virtual]

Function to render the histogram.

Parameters:

data pointer to the vector/scalar to be mapped on histogram

Implements **IRenderer** (p. 161).

13.5.3.4 void CHistogram::histoPreset (const DOMNode * *node*) [private]

This function parse the xml-file to extract the histogram presets.

Parameters:

node pointer to parent DOM node

13.5.3.5 void CHistogram::canvasPreset (const DOMNode * *node*) [private]

This function parse the xml-file to extract the canvas presets.

Parameters:

node pointer to parent DOM node

13.5.3.6 void CHistogram::setCanvas (void) [private]

Construct the root canvas.

13.5.3.7 void CHistogram::fillHisto (double *data*) [private]

Function to fill the histogram.

Parameters:

data pointer to the vector/scalar to be mapped on histogram

Todo

Insert an entire buffer like image histogram

13.5.4 Member Data Documentation

13.5.4.1 DOMNode* CHistogram::hnode_ [private]

Histogram presets DOM node

13.5.4.2 DOMNode* CHistogram::cnode_ [private]

Canvas presets DOM node

13.5.4.3 std::string CHistogram::name_ [private]

Histogram name

13.5.4.4 std::string CHistogram::title_ [private]

Histogram title

13.5.4.5 unsigned long CHistogram::nBins_ [private]

Number of bins

13.5.4.6 double CHistogram::min_ [private]

x min value

13.5.4.7 double CHistogram::max_ [private]

x max value

13.5.4.8 std::string CHistogram::cTitle_ [private]

Canvas title

13.5.4.9 unsigned int CHistogram::cWidth_ [private]

Canvas width

13.5.4.10 unsigned int CHistogram::cHeight_ [private]

Canvas height

13.5.4.11 TH1D* CHistogram::histo_ [private]

Pointer to a ROOT histogram (Double type)

13.5.4.12 TCanvas* CHistogram::canv_ [private]

Pointer to a ROOT canvas

13.5.4.13 var_type CHistogram::vType_ [private]

Histogram variable type

13.5.4.14 unsigned int CHistogram::counter [private]

Counter for time

13.5.4.15 unsigned int CHistogram::version [private]

Graph version

13.5.4.16 CXMLParser* CHistogram::parser_ [private]

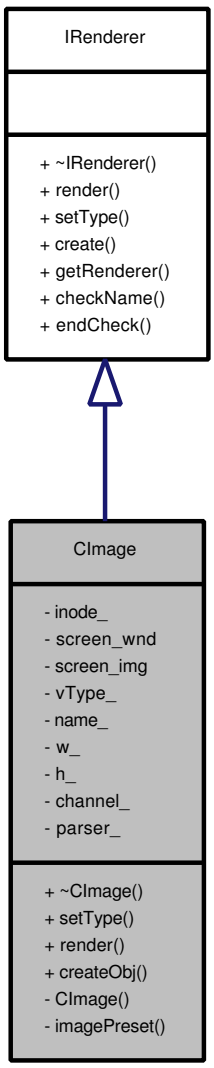
Pointer to Xml parser

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

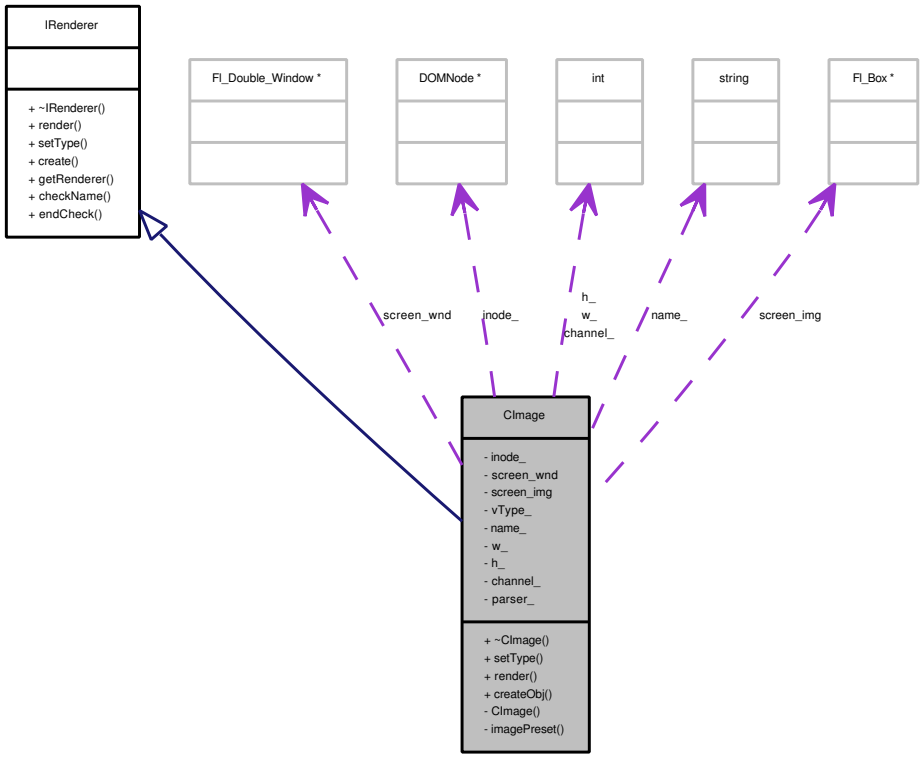
- **CHistogram.h**
- **CHistogram.cpp**

13.6 CImage Class Reference

Implementation of **IRenderer** (p. 160) interface to render a raw image using FLTK libraries. Inheritance diagram for CImage:



Collaboration diagram for CImage:



Public Member Functions

- void **setType** (**var_type** vType)
- void **render** (void *data, unsigned long size)

Static Public Member Functions

- static **IRenderer** * **createObj** (const void *node)

Private Member Functions

- **CImage** (const DOMNode *node)
- void **imagePreset** (const DOMNode *node)

Private Attributes

- DOMNode * **inode_**
- Fl_Double_Window * **screen_wnd**
- Fl_Box * **screen_img**
- **var_type vType_**
- std::string **name_**
- unsigned int **w_**

- unsigned int **h_**
- unsigned int **channel_**
- **CXMLParser** * **parser_**

13.6.1 Detailed Description

Todo

Add possible compressions and formats

Todo

Add Targets (one or more)

Todo

Data type switching

13.6.2 Constructor & Destructor Documentation

13.6.2.1 CImage::~CImage ()

13.6.2.2 CImage::CImage (const DOMNode * *node*) [private]

Constructor: it intializes the windows and the screen (FLTK libraries).

Parameters:

node pointer to parent DOM node

13.6.3 Member Function Documentation

13.6.3.1 IRenderer * CImage::createObj (const void * *node*) [static]

This function, used by the object factory, return an istance of **CImage** (p. 87) class.

Parameters:

node pointer to a generic node (in this case DOMNode) where extracts the image presets.

13.6.3.2 void CImage::setType (var_type *vType*) [virtual]

Function to set the data type.

Parameters:

vType data type

Implements **IRenderer** (p. 161).

13.6.3.3 void CImage::render (void * *data*, unsigned long *size*) [virtual]

Function to rendet the image.

Parameters:

data pointer to the raw image vector

Implements **IRenderer** (p. 161).

13.6.3.4 void CImage::imagePreset (const DOMNode * *node*) [private]

This function parse the xml-file to extract the image presets.

Parameters:

node pointer to parent DOM node

13.6.4 Member Data Documentation

13.6.4.1 DOMNode* CImage::inode_ [private]

Image presets DOM node

13.6.4.2 Fl_Double_Window* CImage::screen_wnd [private]

Pointer to the window

13.6.4.3 Fl_Box* CImage::screen_img [private]

Pointer to the screen

13.6.4.4 var_type CImage::vType_ [private]

Image variable type

13.6.4.5 std::string CImage::name_ [private]

Window caption

13.6.4.6 unsigned int CImage::w_ [private]

Image width

13.6.4.7 unsigned int CImage::h_ [private]

Image height

13.6.4.8 unsigned int CImage::channel_ [private]

Number of color channel of the image

13.6.4.9 CXMLParser* CImage::parser_ [private]

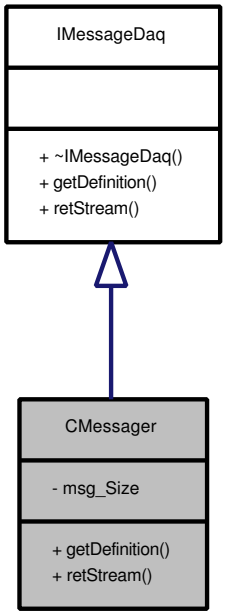
Pointer to Xml parser

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

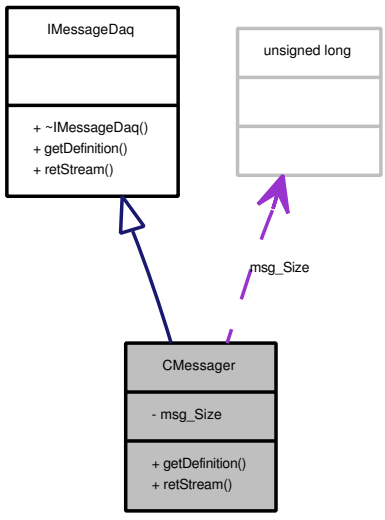
- **CImage.h**
- **CImage.cpp**

13.7 CMessager Class Reference

Implementation of **IMessageDaq** (p. 158) 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**

13.7.1 Member Function Documentation

13.7.1.1 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. 267)

Implements **IMessageDaq** (p. 158).

13.7.1.2 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. 267)

Implements **IMessageDaq** (p. 159).

13.7.2 Member Data Documentation

13.7.2.1 unsigned long CMessager::msg_Size [private]

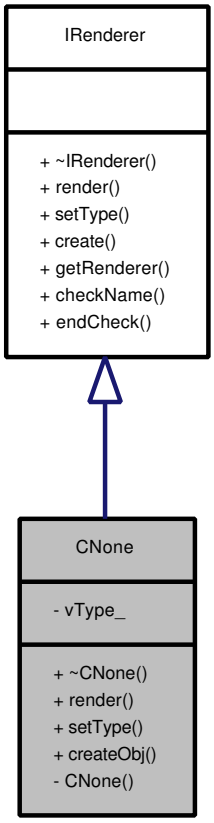
Size of a single message

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

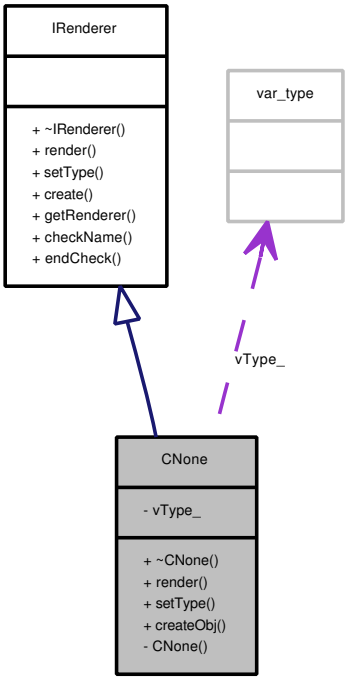
- **CMessager.h**
- **CMessager.cpp**

13.8 CNone Class Reference

Implementation of **IRenderer** (p. 160) interface to handle data with no semantic. This class is implemented to discrard not important character in the stream (like separator). Inheritance diagram for CNone:



Collaboration diagram for CNone:



Public Member Functions

- void **render** (void *data, unsigned long size)
- void **setType** (**var_type** vType)

Private Attributes

- **var_type vType_**

13.8.1 Constructor & Destructor Documentation

13.8.1.1 CNone::~CNone (void)

13.8.1.2 CNone::CNone (const DOMNode * *node*) [private]

13.8.2 Member Function Documentation

13.8.2.1 IRenderer * CNone::createObj (const void * *node*) [static]

13.8.2.2 void CNone::render (void * *data*, unsigned long *size*) [virtual]

This method render data.

Parameters:

data data to be rendered (input)

Implements **IRenderer** (p. 161).

13.8.2.3 void CNone::setType (var_type *vType*) [virtual]

This method set the data type inside the renderer object.

Parameters:

vType data type to set

Implements **IRenderer** (p. 161).

13.8.3 Member Data Documentation

13.8.3.1 var_type CNone::vType_ [private]

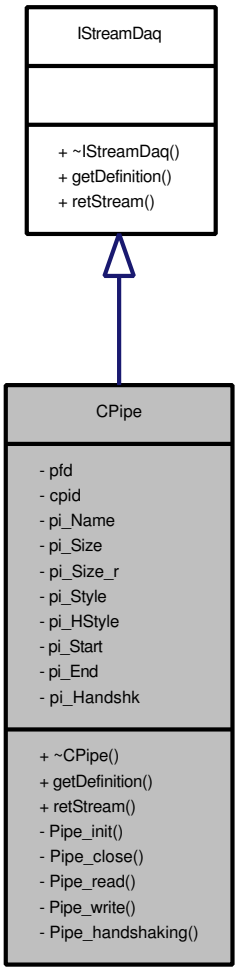
Variable type

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

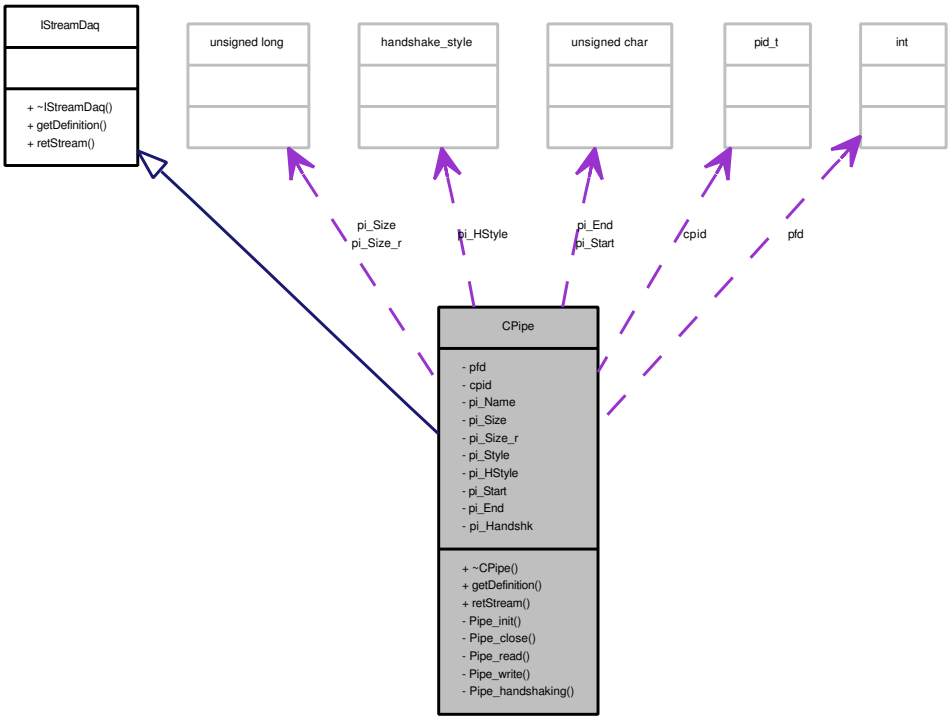
- **CNone.h**
- **CNone.cpp**

13.9 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. 171) 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**

13.9.1 Constructor & Destructor Documentation

13.9.1.1 CPipe::~CPipe ()

CPipe (p. 97) destructor.

13.9.2 Member Function Documentation

13.9.2.1 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. 163).

13.9.2.2 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. 267)

Implements **IStreamDaq** (p. 164).

13.9.2.3 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. 267)

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

Closing the connection to the pipe.

13.9.2.5 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. 267)

13.9.2.6 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. 267)

13.9.2.7 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. 267)

13.9.3 Member Data Documentation

13.9.3.1 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.

13.9.3.2 pid_t CPipe::cpid [private]

Pipe handle

13.9.3.3 char* CPipe::pi_Name [private]

Pipe name

13.9.3.4 unsigned long CPipe::pi_Size [private]

Packet total size

13.9.3.5 unsigned long CPipe::pi_Size_r [private]

Received packet total size

13.9.3.6 s_read_style CPipe::pi_Style [private]

Reading style

13.9.3.7 handshake_style CPipe::pi_HStyle [private]

Handsaking style

13.9.3.8 unsigned char CPipe::pi_Start [private]

Start character

13.9.3.9 unsigned char CPipe::pi_End [private]

End character

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

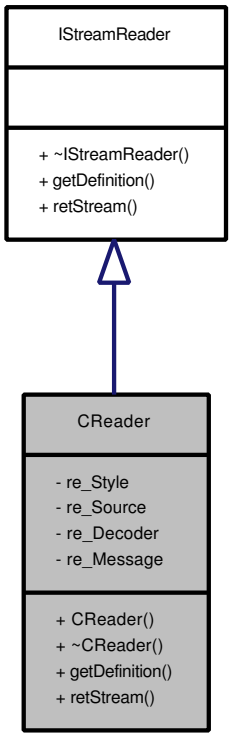
Handshaking packets

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

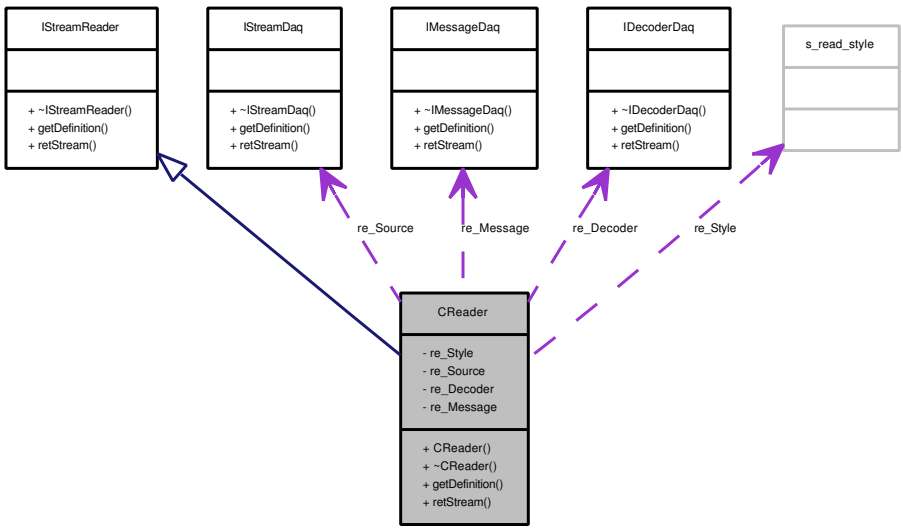
- **CPipe.h**
- **CPipe.cpp**

13.10 CReader Class Reference

Implemenation for **IStreamReader** (p. 165) interface to read, decode and divide packet to be passed to formatter. This class uses instantiation of the classes **IStreamDaq** (p. 163), **IDecoderDaq** (p. 154), **IMessageDaq** (p. 158). 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**

13.10.1 Constructor & Destructor Documentation

13.10.1.1 CReader::CReader ()

CReader (p. 102) constructor.

13.10.1.2 CReader::~CReader ()

CReader (p. 102) destructor.

13.10.2 Member Function Documentation

13.10.2.1 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. 267)

Todo

usb

Todo

stdin

Implements **IStreamReader** (p. 165).

13.10.2.2 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. 267)

Implements **IStreamReader** (p. 166).

13.10.3 Member Data Documentation

13.10.3.1 s_read_style CReader::re_Style [private]

Style of reading.

13.10.3.2 IStreamDaq* CReader::re_Source [private]

Source from wich data is read

13.10.3.3 IDecoderDaq* CReader::re_Decoder [private]

Type of decoder choosen

13.10.3.4 IMessageDaq* CReader::re_Message [private]

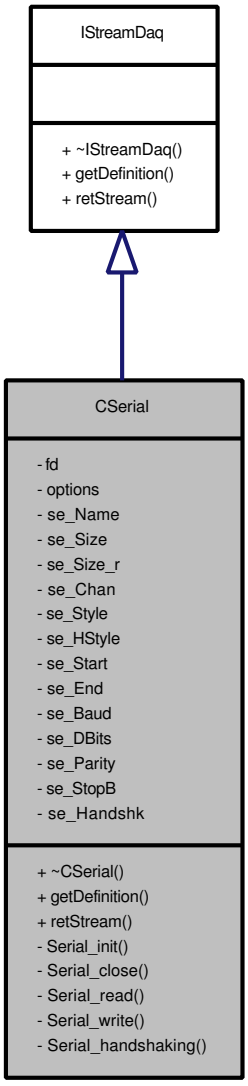
Class to transform pkts in msgs

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

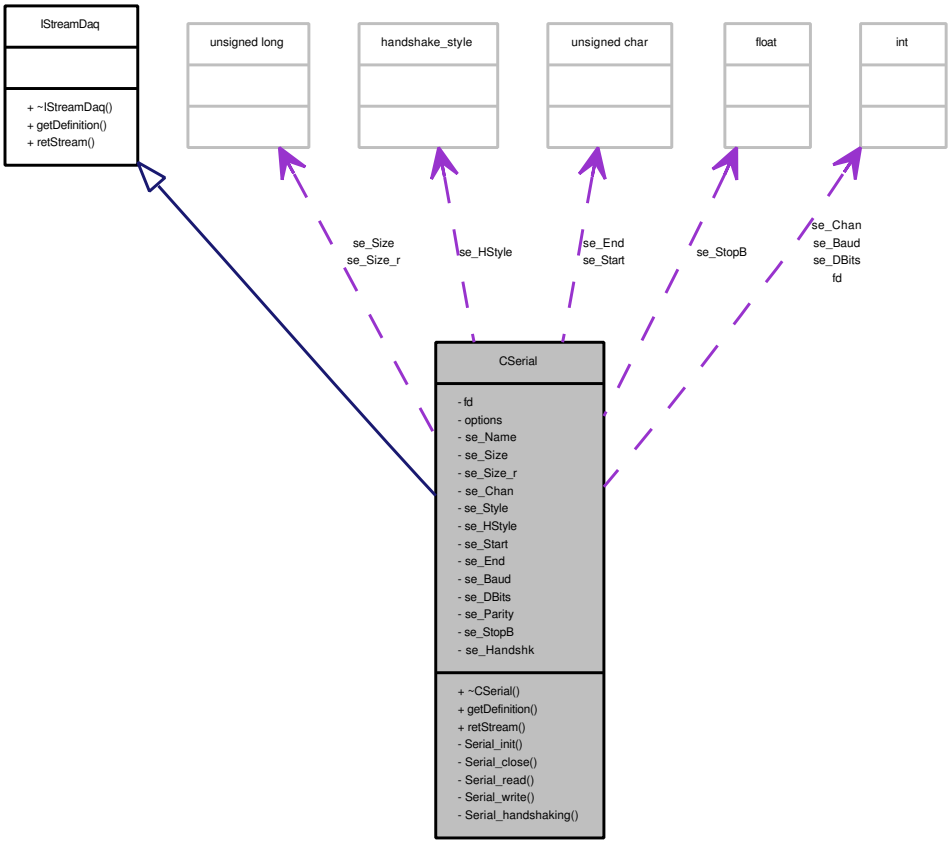
- **CReader.h**
- **CReader.cpp**

13.11 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. 171) 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**

13.11.1 Constructor & Destructor Documentation

13.11.1.1 CSerial::~CSerial ()

CSerial (p. 105) destructor.

13.11.2 Member Function Documentation

13.11.2.1 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. 163).

13.11.2.2 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. 267)

Implements **IStreamDaq** (p. 164).

13.11.2.3 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. 267)

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

Closing of serial port.

13.11.2.5 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. 267)

13.11.2.6 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. 267)

13.11.2.7 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. 267)

13.11.3 Member Data Documentation

13.11.3.1 int CSerial::fd [private]

Handle to serial port

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

Posix structure for serial communication

13.11.3.3 char* CSerial::se_Name [private]

Serial name

13.11.3.4 unsigned long CSerial::se_Size [private]

Packet total size

13.11.3.5 unsigned long CSerial::se_Size_r [private]

Received packet total size

13.11.3.6 unsigned int CSerial::se_Chan [private]

Logical channel

13.11.3.7 s_read_style CSerial::se_Style [private]

Reading style

13.11.3.8 handshake_style CSerial::se_HStyle [private]

Handsaking style

13.11.3.9 unsigned char CSerial::se_Start [private]

Start character

13.11.3.10 unsigned char CSerial::se_End [private]

End character

13.11.3.11 unsigned int CSerial::se_Baud [private]

Baud rate

13.11.3.12 unsigned int CSerial::se_DBits [private]

Number of data bits

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

Parity bit

13.11.3.14 float CSerial::se_StopB [private]

Number of stop bits

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

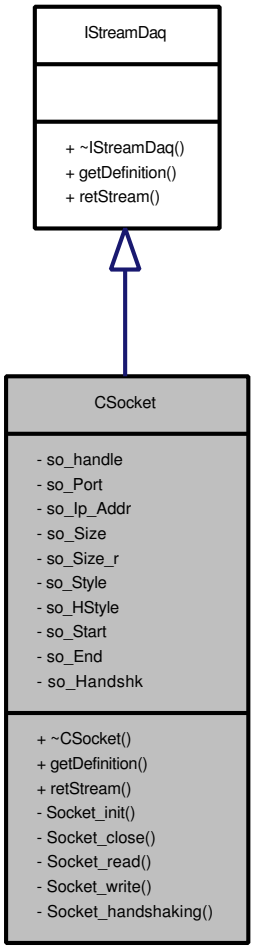
Handshaking packets to start communication

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

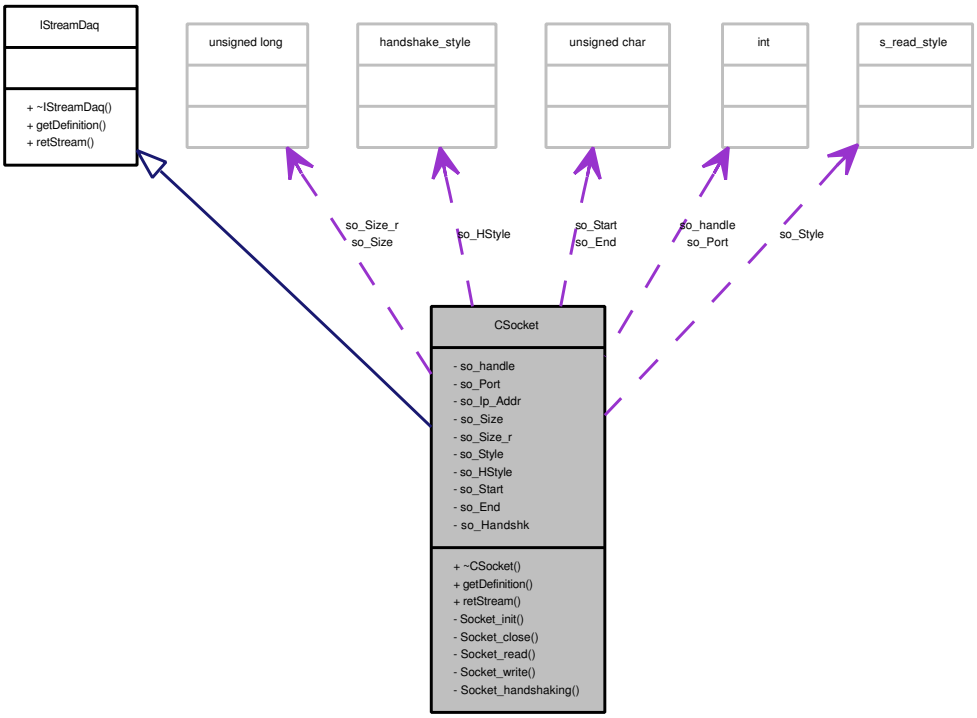
- **CSerial.h**
- **CSerial.cpp**

13.12 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. 171) 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**

13.12.1 Constructor & Destructor Documentation

13.12.1.1 CSocket::~CSocket ()

CPipe (p. 97) destructor.

13.12.2 Member Function Documentation

13.12.2.1 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. 163).

13.12.2.2 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. 267)

Implements **IStreamDaq** (p. 164).

13.12.2.3 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. 267)

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

Closing the socket connection.

13.12.2.5 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. 267)

13.12.2.6 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. 267)

13.12.2.7 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. 267)

13.12.3 Member Data Documentation

13.12.3.1 int CSocket::so_handle [private]

Socket handle

13.12.3.2 unsigned int CSocket::so_Port [private]

Port number

13.12.3.3 char* CSocket::so_Ip_Addr [private]

Server IP address

13.12.3.4 unsigned long CSocket::so_Size [private]

Packet total size

13.12.3.5 unsigned long CSocket::so_Size_r [private]

Received packet total size

13.12.3.6 s_read_style CSocket::so_Style [private]

Reading style

13.12.3.7 handshake_style CSocket::so_HStyle [private]

Handsaking style

13.12.3.8 unsigned char CSocket::so_Start [private]

Start character

13.12.3.9 unsigned char CSocket::so_End [private]

End character

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

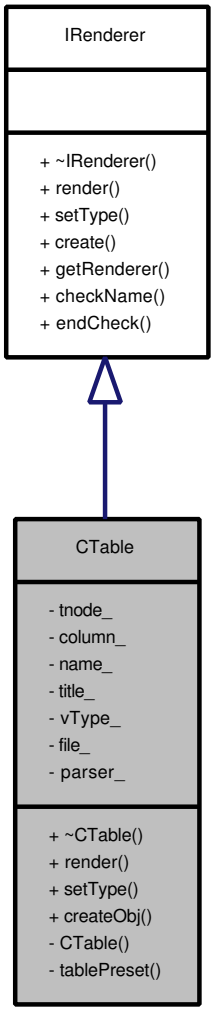
Handshaking packets

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

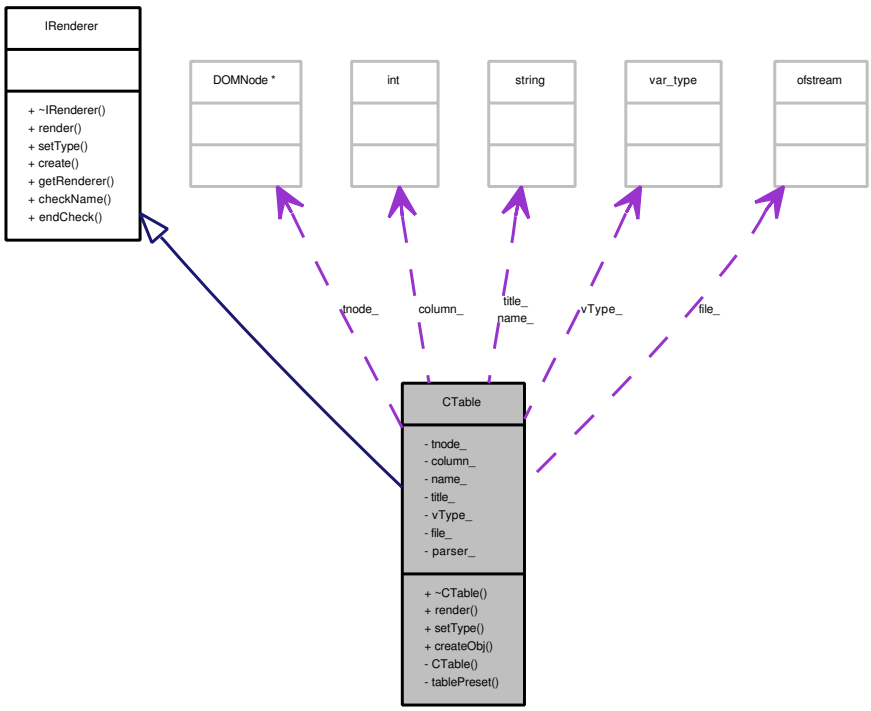
- **CSocket.h**
- **CSocket.cpp**

13.13 CTable Class Reference

Implementation of **IRenderer** (p. 160) interface to write a table inside a file. Inheritance diagram for CTable:



Collaboration diagram for CTable:



Public Member Functions

- void **render** (void *data, unsigned long size)
- void **setType** (**var_type** vType)

Static Public Member Functions

- static **IRenderer** * **createObj** (const void *node)

Private Member Functions

- **CTable** (const DOMNode *node)

Private Attributes

- **CXMLParser** * **parser_**

13.13.1 Constructor & Destructor Documentation

13.13.1.1 CTable::~CTable (void)

13.13.1.2 CTable::CTable (const DOMNode * *node*) [private]

Constructor: it intializes the file handler and the column counter.

Parameters:

node pointer to parent DOM node

13.13.2 Member Function Documentation

13.13.2.1 IRenderer * CTable::createObj (const void * *node*) [static]

This function, used by the object factory, return an istance of **CTable** (p. 117) class.

Parameters:

node pointer to a generic node (in this case DOMNode) where extracts the table presets.

13.13.2.2 void CTable::render (void * *data*, unsigned long *size*) [virtual]

Function to render the histogram.

Parameters:

data pointer to the vector/scalar to be mapped on histogram

Implements **IRenderer** (p. 161).

13.13.2.3 void CTable::setType (var_type *vType*) [virtual]

Function to set the data type.

Parameters:

data pointer to the vector/scalar to be mapped on table

Implements **IRenderer** (p. 161).

13.13.2.4 void CTable::tablePreset (const DOMNode * *node*) [private]

13.13.3 Member Data Documentation

13.13.3.1 DOMNode* CTable::tnode_ [private]

13.13.3.2 unsigned int CTable::column_ [private]

13.13.3.3 std::string CTable::name_ [private]

13.13.3.4 std::string CTable::title_ [private]

13.13.3.5 var_type CTable::vType_ [private]

13.13.3.6 std::ofstream CTable::file_ [private]

13.13.3.7 CXMLParser* CTable::parser_ [private]

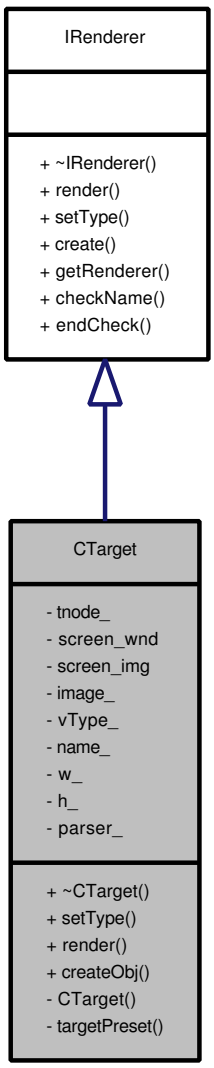
Pointer to Xml parser

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

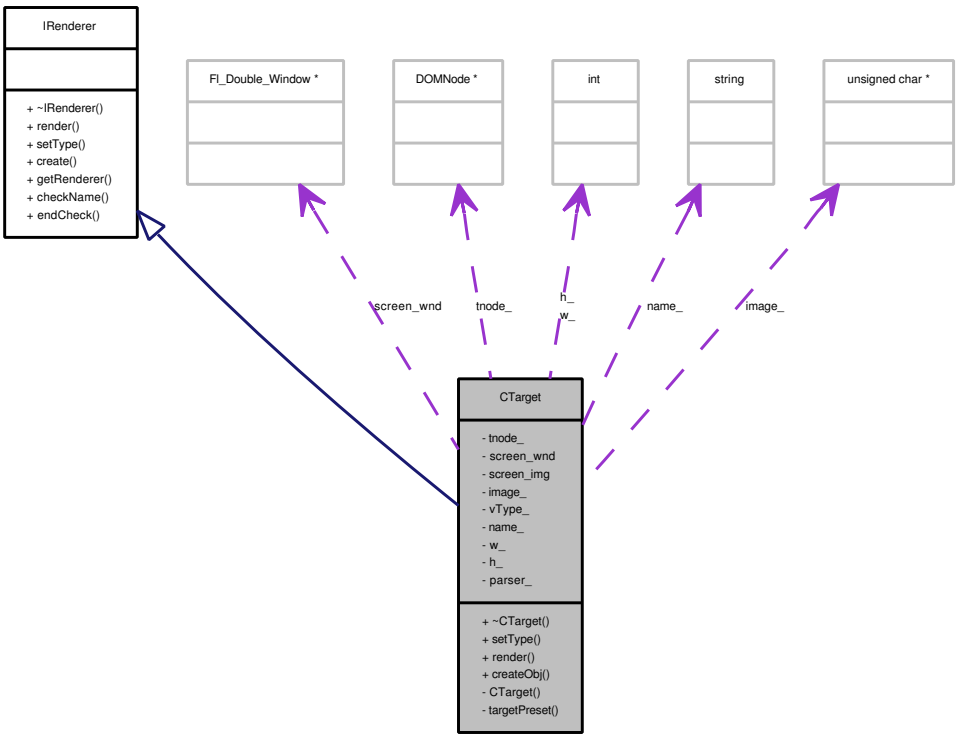
- **CTable.h**
- **CTable.cpp**

13.14 CTarget Class Reference

Implementation of **IRenderer** (p. 160) interface to render a black screen to show the position of a target using FLTK libraries. Inheritance diagram for CTarget:



Collaboration diagram for CTarget:



Public Member Functions

- void **setType** (**var_type** vType)
- void **render** (void *data, unsigned long size)

Static Public Member Functions

- static **IRenderer** * **createObj** (const void *node)

Private Member Functions

- **CTarget** (const DOMNode *node)
- void **targetPreset** (const DOMNode *node)

Private Attributes

- DOMNode * **tnode_**
- Fl_Double_Window * **screen_wnd**
- Fl_Box * **screen_img**
- unsigned char * **image_**
- **var_type vType_**
- std::string **name_**
- unsigned int **w_**

- unsigned int **h_**
- **CXMLParser** ∗ **parser_**

13.14.1 Constructor & Destructor Documentation

13.14.1.1 CTarget::∼CTarget ()

13.14.1.2 CTarget::CTarget (const DOMNode ∗ *node*) [private]

Constructor: it intializes the windows and the screen (FLTK libraries).

Parameters:

node pointer to parent DOM node

13.14.2 Member Function Documentation

13.14.2.1 IRenderer ∗ CTarget::createObj (const void ∗ *node*) [static]

This function, used by the object factory, return an istance of **CTarget** (p. 121) class.

Parameters:

node pointer to a generic node (in this case DOMNode) where extracts the Target presets.

13.14.2.2 void CTarget::setType (var_type *vType*) [virtual]

Function to set the data type.

Parameters:

vType data type

Implements **IRenderer** (p. 161).

13.14.2.3 void CTarget::render (void ∗ *data*, unsigned long *size*) [virtual]

Function to render the Target.

Parameters:

data pointer to 2D target vector

Implements **IRenderer** (p. 161).

13.14.2.4 void CTarget::targetPreset (const DOMNode ∗ *node*) [private]

This function parse the xml-file to extract the target presets.

Parameters:

node pointer to parent DOM node

13.14.3 Member Data Documentation

13.14.3.1 DOMNode* CTarget::tnode_ [private]

target presets DOM node

13.14.3.2 Fl_Double_Window* CTarget::screen_wnd [private]

Pointer to the window

13.14.3.3 Fl_Box* CTarget::screen_img [private]

Pointer to the screen

13.14.3.4 unsigned char* CTarget::image_ [private]

All 0 vector (black image)

13.14.3.5 var_type CTarget::vType_ [private]

Target variable type

13.14.3.6 std::string CTarget::name_ [private]

Window caption

13.14.3.7 unsigned int CTarget::w_ [private]

Image width

13.14.3.8 unsigned int CTarget::h_ [private]

Image height

13.14.3.9 CXMLParser* CTarget::parser_ [private]

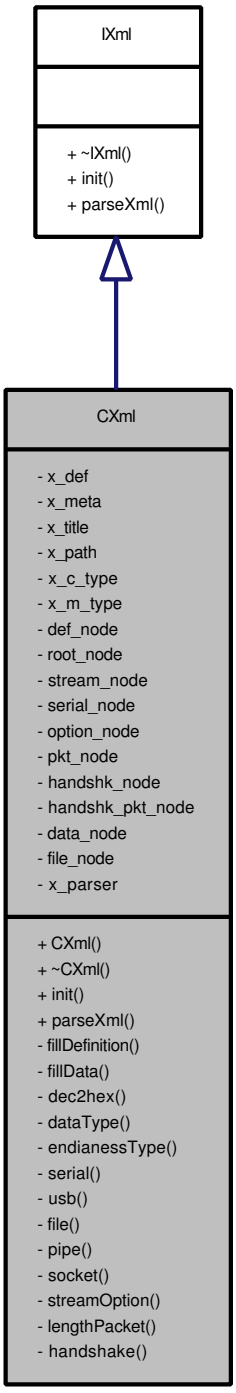
Pointer to Xml parser

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

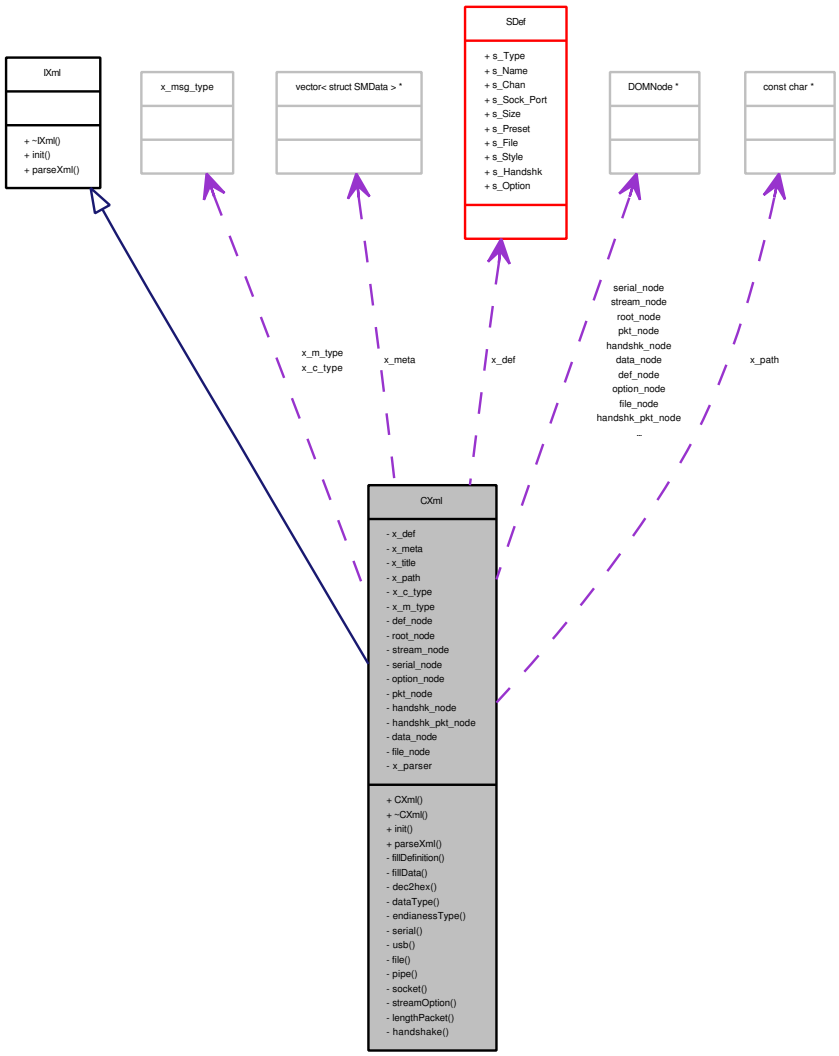
- **CTarget.h**
- **CTarget.cpp**

13.15 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)
- **endianess_type endianessType** (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_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**
- DOMNode * **file_node**
- **CXMLParser** * **x_parser**

13.15.1 Constructor & Destructor Documentation

13.15.1.1 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)

13.15.1.2 CXml::~CXml ()

Destructor.

13.15.2 Member Function Documentation

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

Initialize the parser.

Returns:

This method return one of the possible value from enum

Implements **IXml** (p. 168).

13.15.2.2 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. 168).

13.15.2.3 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. 270)

13.15.2.4 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. 270)

13.15.2.5 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

13.15.2.6 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

13.15.2.7 endianess_type CXml::endianessType (char * *ret*) [private]

Convert a string (char*) in endianess_type.

Parameters:

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

Returns:

return one of the semantic type of endianess_type

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

Fill the preset field of x_def for serial.

Returns:

return one of the x_ret value

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

Fill the preset field of x_def for usb.

Returns:

return one of the x_ret value

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

Fill the preset field of x_def for file.

Returns:

return one of the x_ret value

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

Fill the preset field of x_def for pipe.

Returns:

return one of the x_ret value

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

Fill the preset field of x_def for socket.

Returns:

return one of the x_ret value

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

Fill the structure x_def->s_Option.

See also:

SOption (p. 181)

Returns:

return one of the x_ret value

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

Read from xml tree the size of the packet.

Returns:

return one of the x_ret value

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

Fill the structure x_def->s_Handshk.

See also:

SHandshaking (p. 176)

Returns:

return one of the x_ret value

13.15.3 Member Data Documentation

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

SDef (p. 171) variable to fill with the data parsed from xml file

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

Metadata vector

13.15.3.3 string* CXml::x_title [private]

Title

13.15.3.4 const char* CXml::x_path [private]

Path of xml file

13.15.3.5 x_msg_type CXml::x_c_type [private]

Character type for special character

See also:

x_msg_type (p. 271)

13.15.3.6 x_msg_type CXml::x_m_type [private]

Character type for handshake msg

See also:

x_msg_type (p. 271)

13.15.3.7 DOMNode* CXml::def_node [private]

DOMDocument pointer

13.15.3.8 DOMNode* CXml::root_node [private]

DOMNode pointer for root node

13.15.3.9 DOMNode* CXml::stream_node [private]

DOMNode pointer for stream node

13.15.3.10 DOMNode* CXml::serial_node [private]

DOMNode pointer for serial presets node

13.15.3.11 DOMNode* CXml::option_node [private]

DOMNode pointer for serial stream option node

13.15.3.12 DOMNode* CXml::pkt_node [private]

DOMNode pointer for packet description

13.15.3.13 DOMNode* CXml::handshk_node [private]

DOMNode pointer for handshaking node

13.15.3.14 DOMNode* CXml::handshk_pkt_node [private]

DOMNode pointer for handshaking packet node

13.15.3.15 DOMNode* CXml::data_node [private]

DOMNode pointer for data node

13.15.3.16 DOMNode* CXml::file_node [private]

DOMNode pointer for file preset node

13.15.3.17 CXMLParser* CXml::x_parser [private]

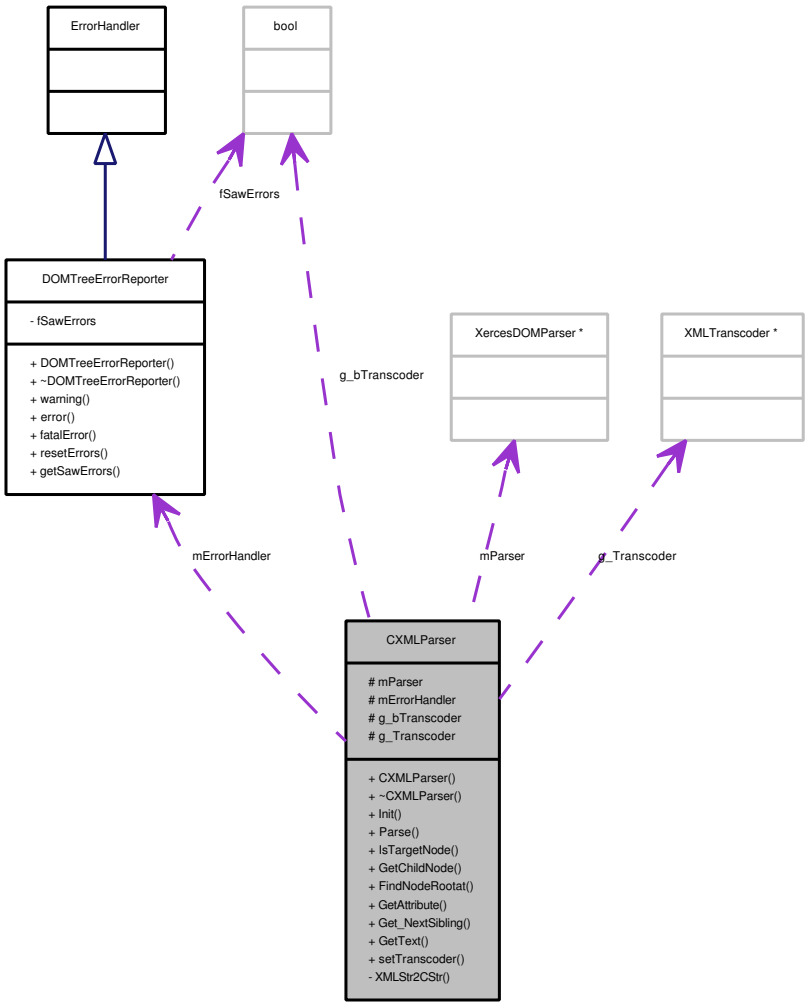
Pointer to Xml parser

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

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

13.16 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)

13.16.1 Constructor & Destructor Documentation

13.16.1.1 CXMLParser::CXMLParser ()

CXMLParser (p. 133) contructor.

13.16.1.2 CXMLParser::~CXMLParser ()

CXMLParser (p. 133) virtual distructor.

13.16.2 Member Function Documentation

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

Method to decode string from XMLCh* to char*.

Parameters:

s Xml string (input)

Returns:

Decoded string

13.16.2.2 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. 269)

13.16.2.3 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. 269)

13.16.2.4 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

13.16.2.5 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

13.16.2.6 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

13.16.2.7 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

13.16.2.8 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

13.16.2.9 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

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

Change string encoder.

Parameters:

codepage Encoding type (input)

13.16.3 Member Data Documentation

13.16.3.1 XercesDOMParser* CXMLParser::mParser [protected]

XercesDOMParser class object

13.16.3.2 DOMTreeErrorReporter* CXMLParser::mErrorHandler [protected]

DOMTreeErrorReporter (p. 148) class object

13.16.3.3 bool CXMLParser::g_bTranscoder [protected]

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

13.16.3.4 XMLTranscoder* CXMLParser::g_Transcoder [protected]

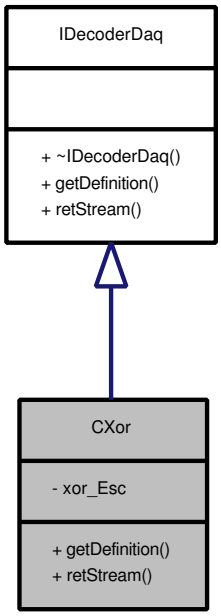
brief Transcoder pointer

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

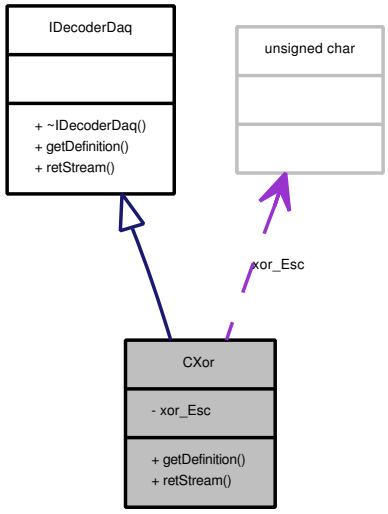
- **CXmlParser.h**
- **CXmlParser.cpp**

13.17 CXor Class Reference

Implementation of **IDecoderDaq** (p. 154) 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**

13.17.1 Member Function Documentation

13.17.1.1 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. 267)

Implements **IDecoderDaq** (p. 154).

13.17.1.2 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. 267)

Implements **IDecoderDaq** (p. 155).

13.17.2 Member Data Documentation

13.17.2.1 unsigned char CXor::xor_Esc [private]

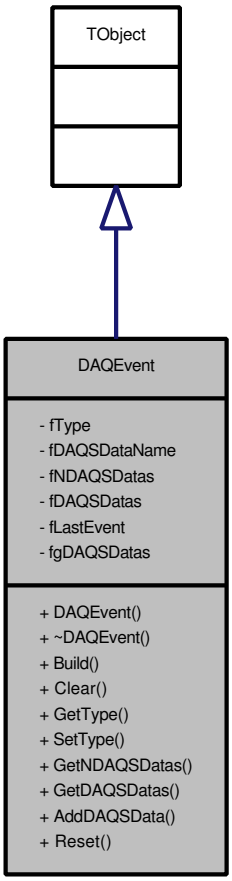
Escape character

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

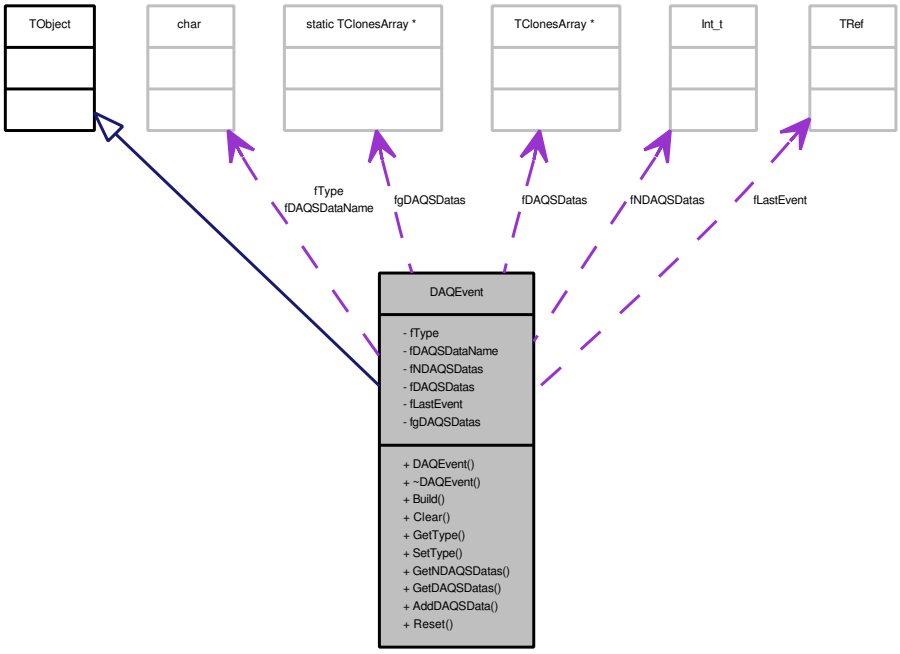
- **CXor.h**
- **CXor.cpp**

13.18 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**

13.18.1 Constructor & Destructor Documentation

13.18.1.1 DAQEvent::DAQEvent ()

13.18.1.2 DAQEvent::~DAQEvent () [virtual]

13.18.2 Member Function Documentation

13.18.2.1 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. 169)

a Pointer to **SData** (p. 169)

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

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

13.18.2.4 char* DAQEvent::GetType ()

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

13.18.2.6 Int_t DAQEvent::GetNDAQSDatas () const

13.18.2.7 TClonesArray* DAQEvent::GetDAQSDatas () const

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

13.18.3 Member Data Documentation

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

String containing the type of the event

13.18.3.2 char* DAQEvent::fDAQSDataName [private]

Run+event number in character format

13.18.3.3 Int_t DAQEvent::fNDAQSDatas [private]

Number of **SData** (p. 169) objects in the event

13.18.3.4 TClonesArray* DAQEvent::fDAQSDatas [private]

Array with all the events stored

13.18.3.5 TRef DAQEvent::fLastEvent [private]

reference pointer to last **SData** (p. 169)

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

static counterpart of fDAQSDatas

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

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

13.19 DAQPayload Struct Reference

Structure that describe the packet payload according with xml file.

13.19.1 Detailed Description

Warning:

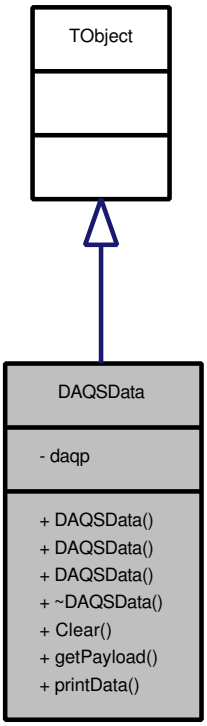
This structure has to be modified everytime the xml file payload description is changed

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

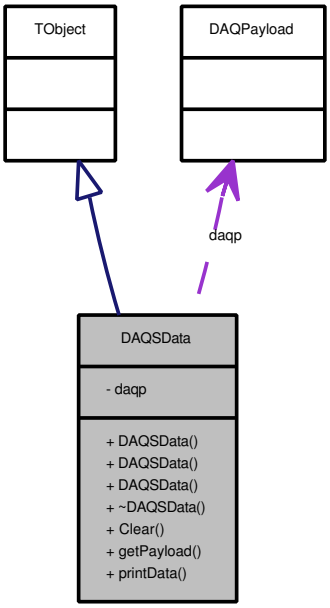
- **DAQPayload.h**

13.20 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**

13.20.1 Detailed Description

Warning:

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

13.20.2 Constructor & Destructor Documentation

13.20.2.1 DAQSData::DAQSData ()

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

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

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

13.20.3 Member Function Documentation

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

13.20.3.2 DAQPayload* DAQSData::getPayload (void)

returns the raw data

13.20.3.3 void DAQSData::printData () const

prints the data as a table

13.20.4 Member Data Documentation

13.20.4.1 DAQPayload DAQSData::daqp [private]

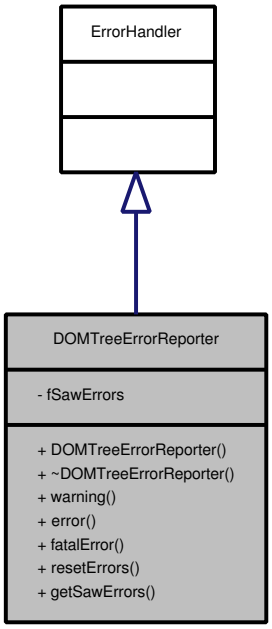
Member raw data

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

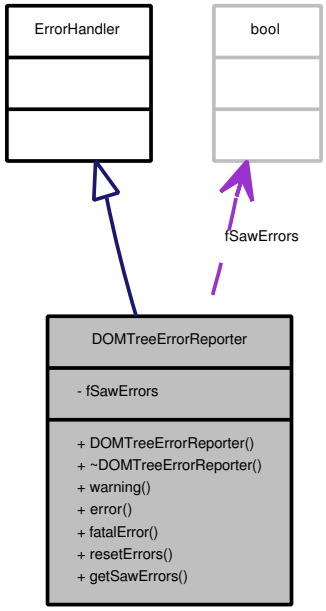
- **DAQSData.h**
- **DAQSData.cxx**

13.21 DOMTreeErrorReporter Class Reference

This class registers as an **ErrorHandler** (p. 151) 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**

13.21.1 Constructor & Destructor Documentation

13.21.1.1 DOMTreeErrorReporter::DOMTreeErrorReporter ()

Constructor.

13.21.1.2 DOMTreeErrorReporter::~DOMTreeErrorReporter ()

Destructor.

13.21.2 Member Function Documentation

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

Warning handler.

Parameters:

toCatch Instance of SAX Parser error handler class

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

Error handler.

Parameters:

toCatch Instance of SAX Parser error handler class

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

Fatal error handler.

Parameters:

toCatch Instance of SAX Parser error handler class

13.21.2.4 void DOMTreeErrorReporter::resetErrors ()

Reset error.

13.21.2.5 bool DOMTreeErrorReporter::getSawErrors () const

Getter methods.

Returns:

True if an error was found

13.21.3 Member Data Documentation

13.21.3.1 bool DOMTreeErrorReporter::fSawErrors [private]

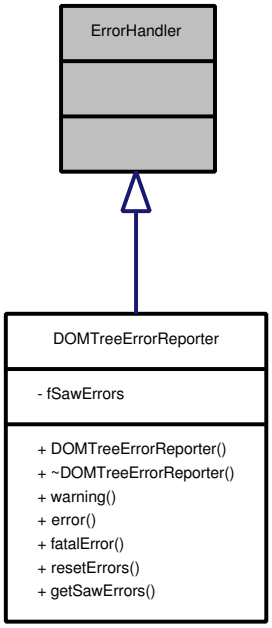
This is set true if we get any errors, and is queryable via a getter method.

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

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

13.22 ErrorHandler Class Reference

Basic interface for SAX error handlers. From the Xerces package. Inheritance diagram for ErrorHandler:

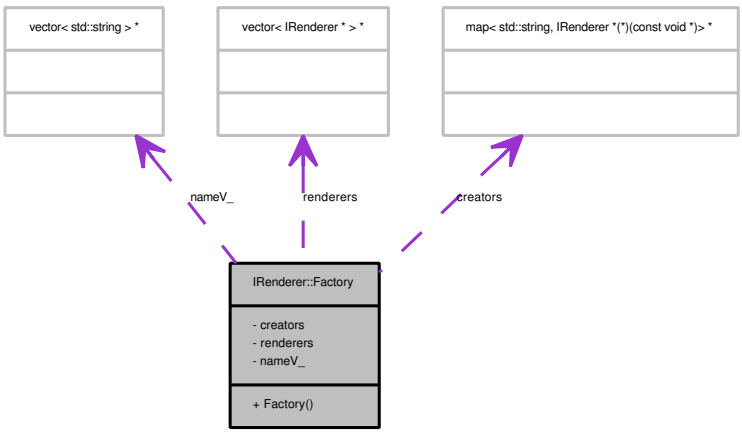


The documentation for this class was generated from the following file:

- **DOMTreeErrorReporter.hpp**

13.23 IRenderer::Factory Class Reference

IRenderer (p. 160) object factory to handle the renderer. Collaboration diagram for IRenderer::Factory:



Public Member Functions

- **Factory** (std::string semantic, **IRenderer** ∗(∗creator)(const void ∗node))

Static Private Attributes

- static std::map< std::string, **IRenderer** ∗(∗)(const void ∗)> ∗ **creators**
- static std::vector< **IRenderer** ∗ > ∗ **renderers**
- static std::vector< std::string > ∗ **nameV_**

13.23.1 Constructor & Destructor Documentation

13.23.1.1 IRenderer::Factory::Factory (std::string *semantic*, IRenderer ∗(∗)(const void ∗node) *creator*)

Contructor: permit to link a string with a method to create a certain renderer using maps.

Parameters:

semantic name of the renderer

creator method to create the respective renderer

13.23.2 Friends And Related Function Documentation

13.23.2.1 friend class IRenderer [friend]

IRenderer (p. 160) has to access to **Factory** (p. 152) privete and static elements.

13.23.3 Member Data Documentation

13.23.3.1 std::map< std::string, IRenderer *(*)(const void *)> * IRenderer::Factory::creators [static, private]

Renderer creator map

13.23.3.2 std::vector< IRenderer * > * IRenderer::Factory::renderers [static, private]

Renderer object vector

13.23.3.3 std::vector< std::string > * IRenderer::Factory::nameV_ [static, private]

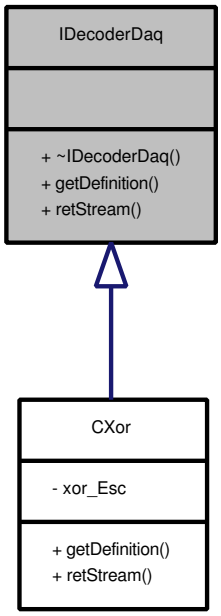
Renderer name vector

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

- **IRenderer.h**
- **IRenderer.cpp**

13.24 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. 102) 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

13.24.1 Constructor & Destructor Documentation

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

13.24.2 Member Function Documentation

13.24.2.1 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. 267)

Implemented in **CXor** (p. 139).

13.24.2.2 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. 267)

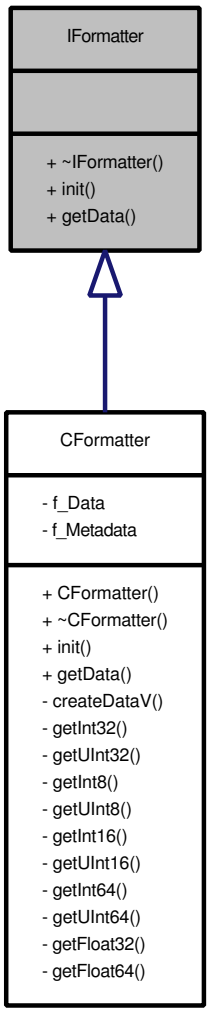
Implemented in **CXor** (p. 139).

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

- **IDecoderDaq.h**

13.25 IFormatter Class Reference

Interface for data formatter. Inheritance diagram for IFormatter:



Public Member Functions

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

13.25.1 Constructor & Destructor Documentation

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

13.25.2 Member Function Documentation

13.25.2.1 virtual f_ret IFormatter::init (std::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. 243)

Implemented in **CFormatter** (p. 70).

13.25.2.2 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. 165) (input)

Returns:

One of the possible value of f_ret

See also:

f_ret (p. 243)

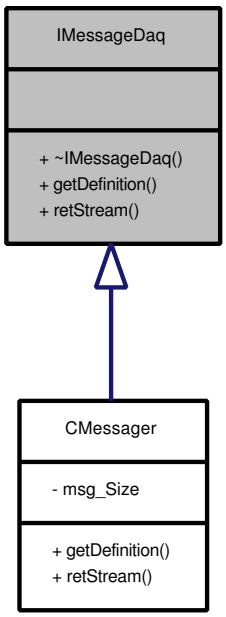
Implemented in **CFormatter** (p. 70).

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

- **IFormatter.h**

13.26 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. 102) 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

13.26.1 Constructor & Destructor Documentation

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

13.26.2 Member Function Documentation

13.26.2.1 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. 267)

Implemented in **CMessager** (p. 93).

13.26.2.2 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. 267)

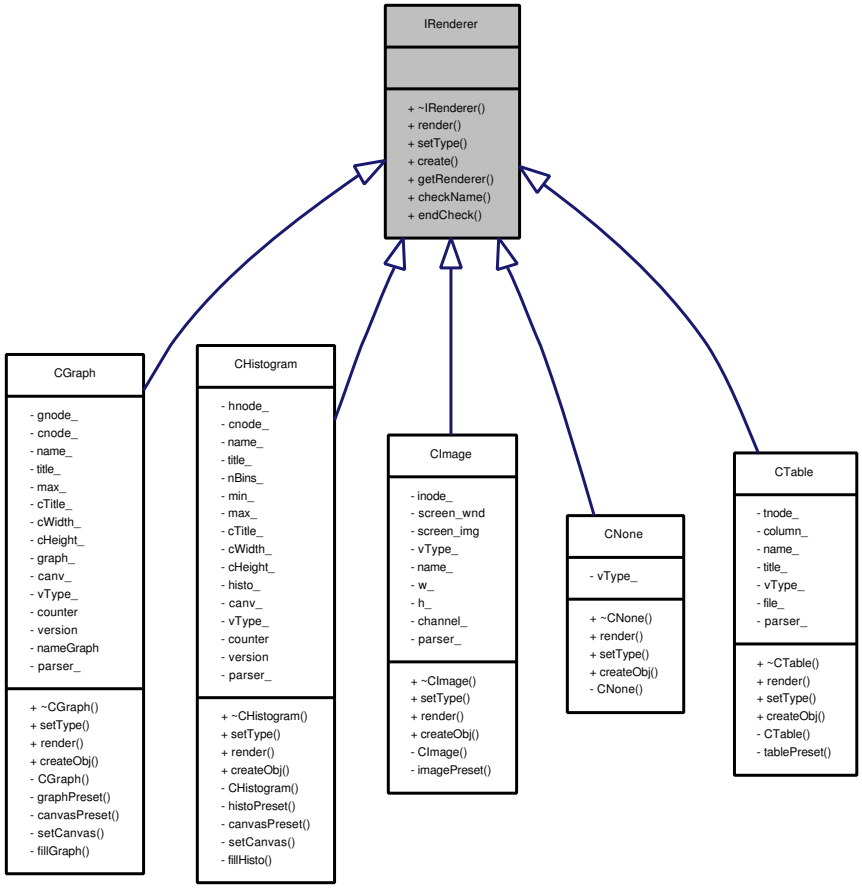
Implemented in **CMessager** (p. 93).

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

- **IMessageDaq.h**

13.27 IRenderer Class Reference

Interface to render data (images, histograms, graphs, tables). Inheritance diagram for IRenderer:



Classes

- class **Factory**

  IRenderer *(p. 160) object factory to handle the renderer.*

Public Member Functions

- virtual void **render** (void ∗data, unsigned long size)=0
- virtual void **setType** (**var_type** vType)=0

Static Public Member Functions

- static void **create** (std::string semantic, const void ∗node, **var_type** vType)
- static **IRenderer** ∗ **getRenderer** (unsigned int i)
- static void **checkName** (std::string name)

13.27.1 Constructor & Destructor Documentation

13.27.1.1 virtual IRenderer::~IRenderer () [virtual]

13.27.2 Member Function Documentation

13.27.2.1 virtual void IRenderer::render (void * *data*, unsigned long *size*) [pure virtual]

This method render data.

Parameters:

data data to be rendered (input)

Implemented in **CGraph** (p. 78), **CHistogram** (p. 84), **CImage** (p. 90), **CNone** (p. 95), **CTable** (p. 119), and **CTarget** (p. 123).

13.27.2.2 virtual void IRenderer::setType (var_type *vType*) [pure virtual]

This method set the data type inside the renderer object.

Parameters:

vType data type to set

Implemented in **CGraph** (p. 78), **CHistogram** (p. 84), **CImage** (p. 89), **CNone** (p. 96), **CTable** (p. 119), and **CTarget** (p. 123).

13.27.2.3 void IRenderer::create (std::string *semantic*, const void * *node*, var_type *vType*) [static]

This method create the renderer, add it to the renderer vector and fix data type in it.

Parameters:

semantic renderer name

node generic node that contains the renderer presets

vType data type to set

13.27.2.4 IRenderer * IRenderer::getRenderer (unsigned int *i*) [static]

This method return the pointer of the i-th renderer.

Parameters:

i renderer position

Returns:

The pointer to the renderer

13.27.2.5 void IRenderer::checkName (std::string *name*) [static]

This method check the correctness of the object name.

Parameters:

name object name to check

13.27.2.6 void IRenderer::endCheck (void) [static]

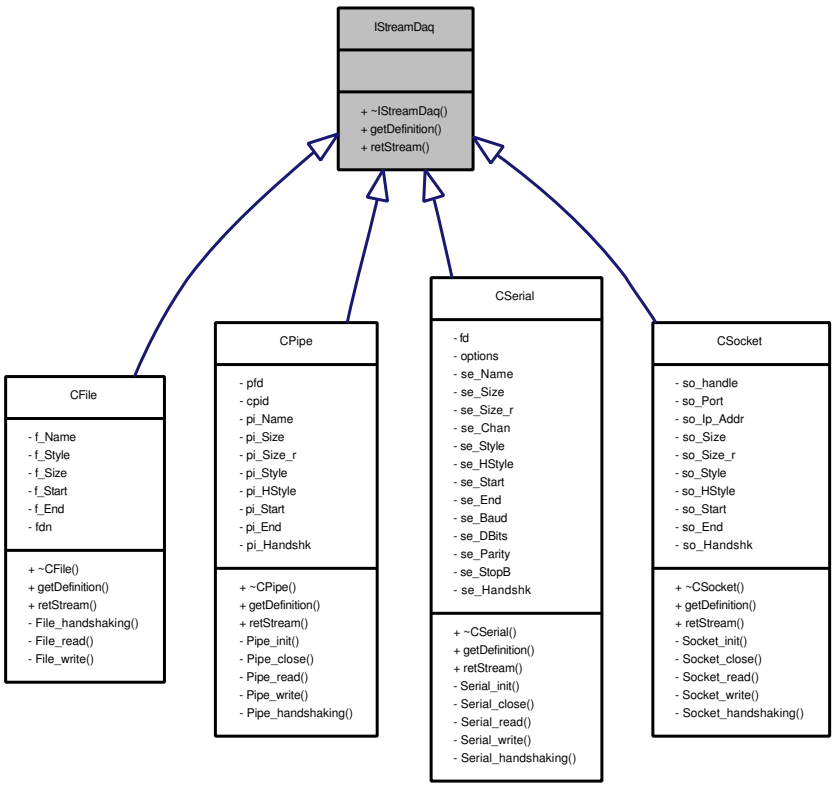
This method delete the name vector.

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

- **IRenderer.h**
- **IRenderer.cpp**

13.28 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. 102) 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

13.28.1 Constructor & Destructor Documentation

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

13.28.2 Member Function Documentation

13.28.2.1 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. 267)

Implemented in **CFile** (p. 65), **CPipe** (p. 99), **CSerial** (p. 107), and **CSocket** (p. 113).

13.28.2.2 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. 267)

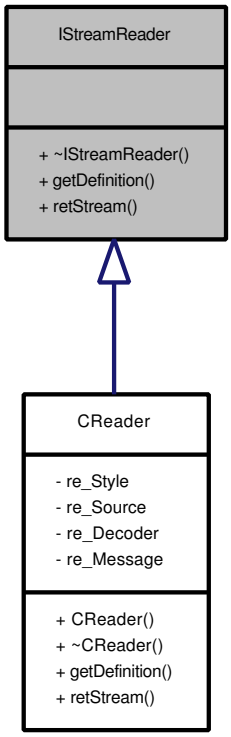
Implemented in **CFile** (p. 65), **CPipe** (p. 99), **CSerial** (p. 107), and **CSocket** (p. 113).

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

- **IStreamDaq.h**

13.29 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

13.29.1 Constructor & Destructor Documentation

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

13.29.2 Member Function Documentation

13.29.2.1 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. 267)

Implemented in **CReader** (p. 103).

13.29.2.2 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. 267)

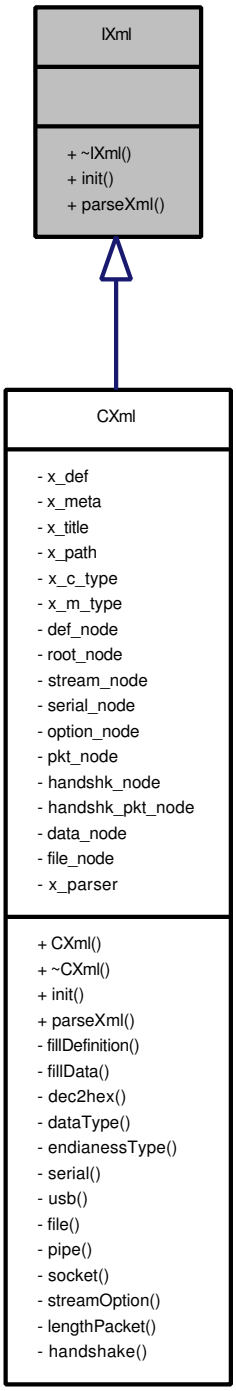
Implemented in **CReader** (p. 103).

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

- **IStreamReader.h**

13.30 IXml Class Reference

Interface for xml parser. Inheritance diagram for IXml:



Public Member Functions

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

13.30.1 Constructor & Destructor Documentation

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

13.30.2 Member Function Documentation

13.30.2.1 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. 270)

Implemented in **CXml** (p. 128).

13.30.2.2 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. 270)

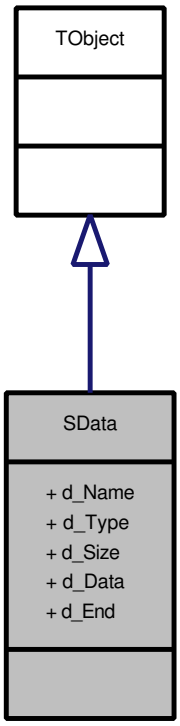
Implemented in **CXml** (p. 128).

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

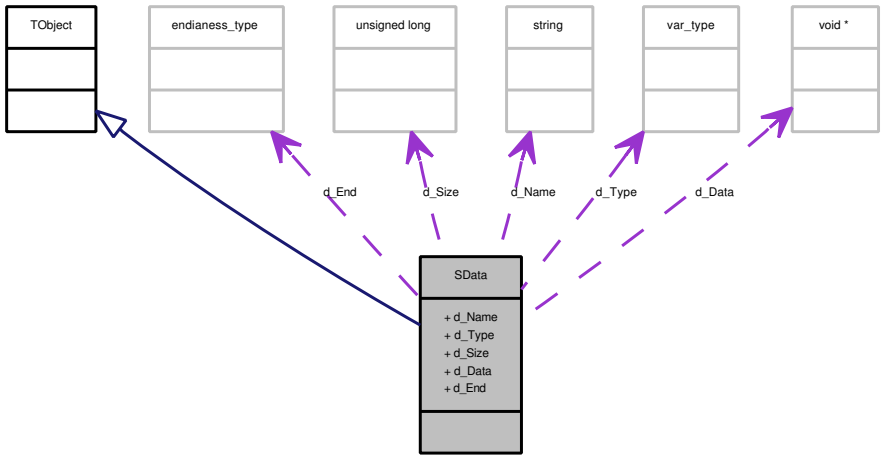
- **IXml.h**

13.31 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**
- **endianess_type d_End**

13.31.1 Member Data Documentation

13.31.1.1 std::string SData::d_Name

Element name

13.31.1.2 var_type SData::d_Type

Type of the element

13.31.1.3 unsigned long SData::d_Size

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

13.31.1.4 void* SData::d_Data

Pointer to a data

13.31.1.5 endianess_type SData::d_End

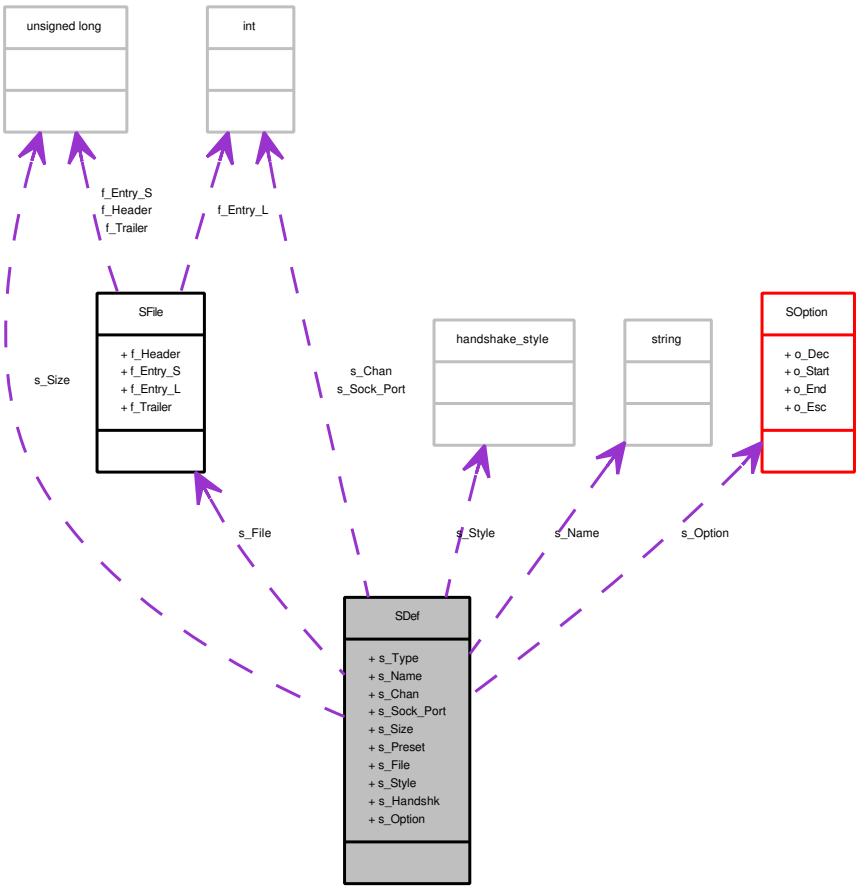
Endianess

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

- **Data_struct.h**

13.32 SDef Struct Reference

Structure for stream definition. Collaboration diagram for SDef:



Public Attributes

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

13.32.1 Member Data Documentation

13.32.1.1 stream_type SDef::s_Type

Stream type

13.32.1.2 std::string SDef::s_Name

Stream name

13.32.1.3 unsigned int SDef::s_Chan

Logical channel number (from stream_info attribute in xml)

13.32.1.4 unsigned int SDef::s_Sock_Port

Socket port number (from stream_info attribute in xml)

13.32.1.5 unsigned long SDef::s_Size

Packet total size

13.32.1.6 struct Srs_232* SDef::s_Preset [read]

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

See also:

Srs_232 (p. 183)

13.32.1.7 struct SFile* SDef::s_File [read]

File structure presets

See also:

SFile (p. 174)

13.32.1.8 handshake_style SDef::s_Style

Handshake style

See also:

handshake_style (p. 226)

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

Handshaking packets to start communication

See also:

SHandshaking (p. 176)

13.32.1.10 struct SOption* SDef::s_Option [read]

Character of syncro and of escape

See also:

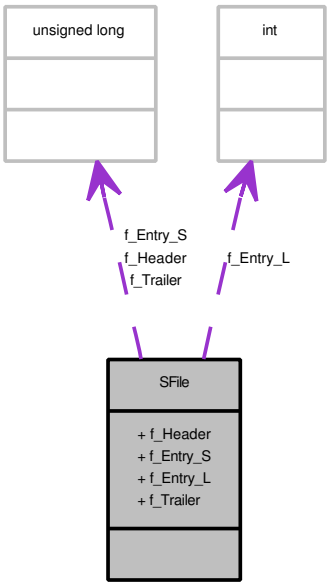
SOption (p. 181)

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

- **DAQ_struct.h**

13.33 SFile Struct Reference

Structure of File. Collaboration diagram for SFile:



Public Attributes

- unsigned long **f_Header**
- unsigned long **f_Entry_S**
- unsigned int **f_Entry_L**
- unsigned long **f_Trailer**

13.33.1 Member Data Documentation

13.33.1.1 unsigned long SFile::f_Header

Header length (in bytes)

13.33.1.2 unsigned long SFile::f_Entry_S

Number of entry (start byte, 0-based)

13.33.1.3 unsigned int SFile::f_Entry_L

Number of entry (length in bytes)

13.33.1.4 unsigned long SFile::f_Trailer

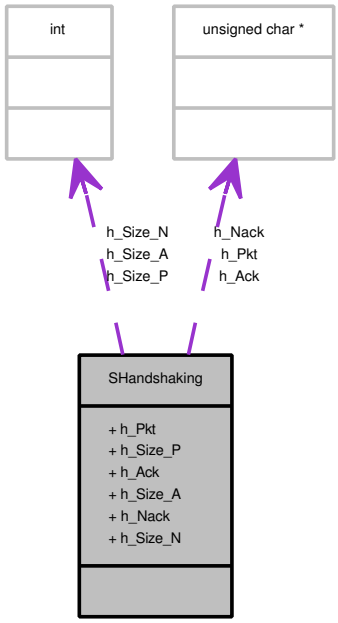
Trailer start (0-based)

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

- DAQ_struct.h

13.34 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**

13.34.1 Member Data Documentation

13.34.1.1 unsigned char* SHandshaking::h_Pkt

Packet to send

13.34.1.2 unsigned int SHandshaking::h_Size_P

Packet to send size

13.34.1.3 unsigned char* SHandshaking::h_Ack

Ack value

13.34.1.4 unsigned int SHandshaking::h_Size_A

Ack size

13.34.1.5 unsigned char* SHandshaking::h_Nack

Nack value

13.34.1.6 unsigned int SHandshaking::h_Size_N

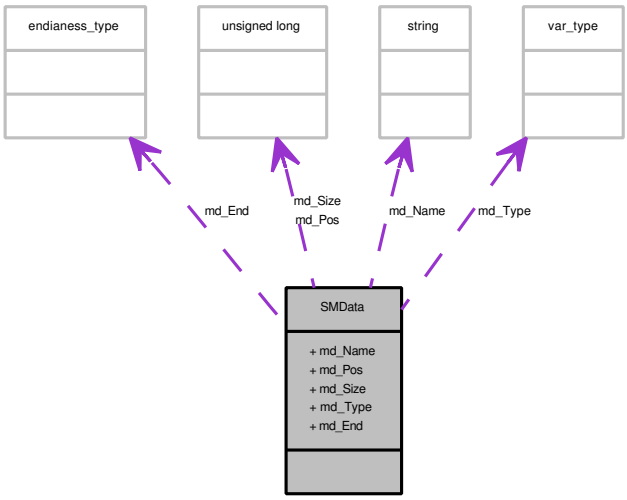
Nack size

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

- **DAQ_struct.h**

13.35 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**
- **endianess_type md_End**

13.35.1 Member Data Documentation

13.35.1.1 std::string SMData::md_Name

Element name

13.35.1.2 unsigned long SMData::md_Pos

Position of the element in the packet (in bytes)

13.35.1.3 unsigned long SMData::md_Size

Size of the element (in bytes)

13.35.1.4 var_type SMData::md_Type

Type of the element

13.35.1.5 endianess_type SMData::md_End

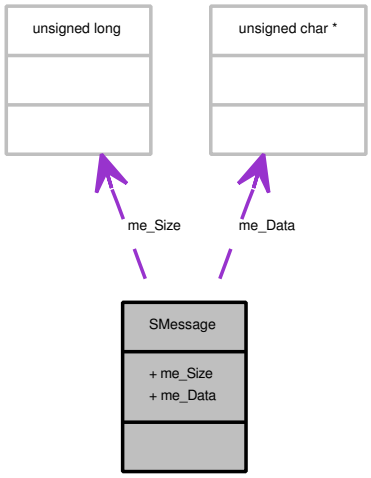
Endianess

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

- **DAQ_struct.h**

13.36 SMessage Struct Reference

Structure of message. Collaboration diagram for SMessage:



Public Attributes

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

13.36.1 Member Data Documentation

13.36.1.1 unsigned long SMessage::me_Size

Size of the message

13.36.1.2 unsigned char* SMessage::me_Data

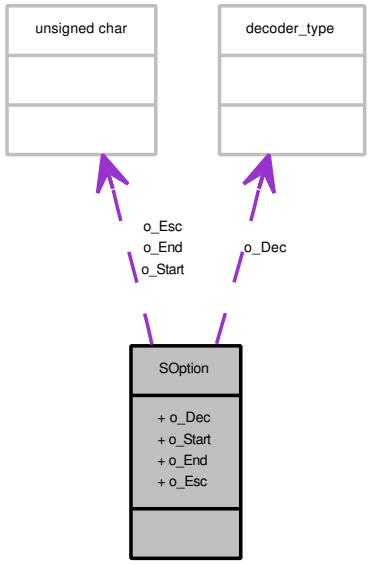
Pointer to the message

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

- **DAQ_struct.h**

13.37 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**

13.37.1 Member Data Documentation

13.37.1.1 decoder_type SOption::o_Dec

Decoder type

13.37.1.2 unsigned char SOption::o_Start

Packet start character

13.37.1.3 unsigned char SOption::o_End

Packet end character

13.37.1.4 unsigned char SOption::o_Esc

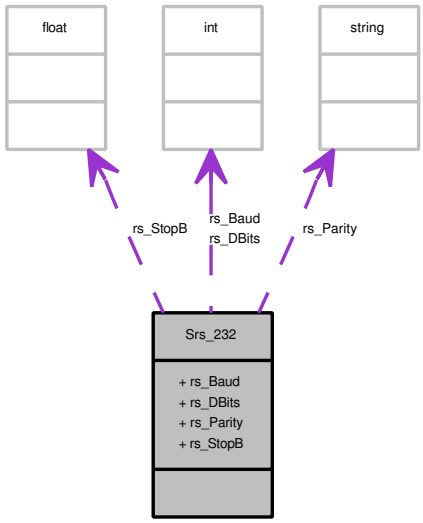
Escape character

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

- **DAQ_struct.h**

13.38 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**

13.38.1 Member Data Documentation

13.38.1.1 unsigned int Srs_232::rs_Baud

Baud rate

13.38.1.2 unsigned int Srs_232::rs_DBits

Number of data bits

13.38.1.3 std::string Srs_232::rs_Parity

Parity bit

13.38.1.4 float Srs_232::rs_StopB

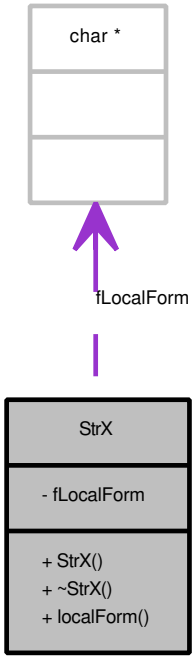
Number of stop bits

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

- **DAQ_struct.h**

13.39 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**

13.39.1 Constructor & Destructor Documentation

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

Constructor.

Parameters:

toTranscode Data to transocode

13.39.1.2 StrX::~StrX ()

Destructor.

13.39.2 Member Function Documentation

13.39.2.1 const char* StrX::localForm () const

Getter methods.

Returns:

Data transcoded

13.39.3 Member Data Documentation

13.39.3.1 char* StrX::fLocalForm [private]

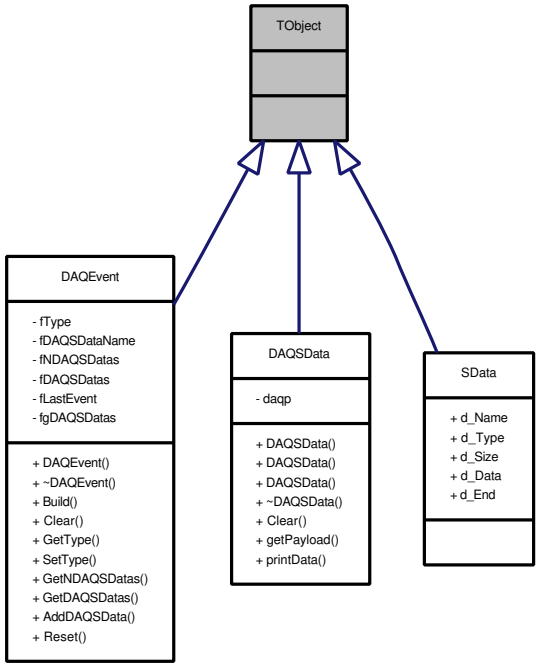
This is the local code page form of the string

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

- **DOMTreeErrorReporter.hpp**

13.40 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:

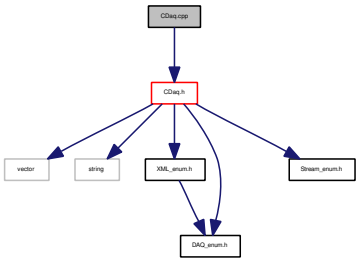
- **DAQEvent.h**

Chapter 14

File Documentation

14.1 CDaq.cpp File Reference

CDaq (p. 59) methods development. Include dependency graph for CDaq.cpp:



Defines

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

14.1.1 Detailed Description

14.1.2 Define Documentation

14.1.2.1 #define XML_PREFIX

Value to be added to CXmlParser return value to be translated to **CDaq** (p. 59) return value

See also:

daq_ret (p. 228)
x_ret (p. 270)

14.1.2.2 #define READER_PREFIX

Value to be added to **CReader** (p. 102) return value to be translated to **CDaq** (p. 59) return value

See also:

daq_ret (p. 228)
s_ret (p. 267)

14.1.2.3 #define FORMATTER_PREFIX

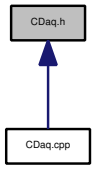
Value to be added to **CFormatter** (p. 68) return value to be translated to **CDaq** (p. 59) return value

See also:

daq_ret (p. 228)
f_ret (p. 243)

14.2 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.

14.2.1 Detailed Description

Date:

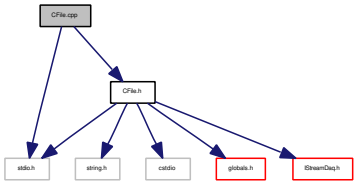
Created on: 31-mag-2009

Author:

Author: Claudio Salvadori, Christian Nastasi and Paolo Pagano

14.3 CFile.cpp File Reference

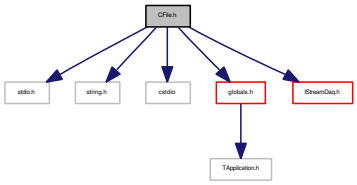
File reading functions development. Include dependency graph for CFile.cpp:



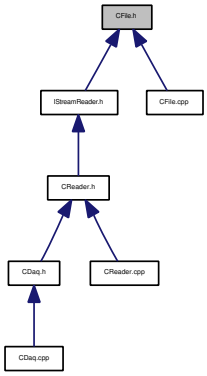
14.3.1 Detailed Description

14.4 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. 171) 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. 171) structure directives.*

14.4.1 Detailed Description

Date:

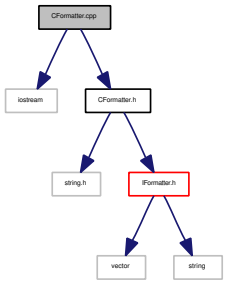
Created on: 22-apr-2009

Author:

Author: Claudio Salvadori

14.5 CFormatter.cpp File Reference

Formatting functions development. Include dependency graph for CFormatter.cpp:



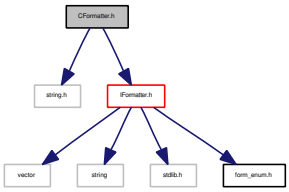
14.5.1 Detailed Description

14.5.2 Define Documentation

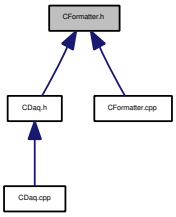
14.5.2.1 #define U_SIZE

14.6 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).

14.6.1 Detailed Description

Date:

Created on: 22-apr-2009

Author:

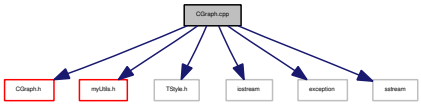
Author: Claudio Salvadori

See also:

SMData (p. 178)

14.7 CGraph.cpp File Reference

Graph class method development. Include dependency graph for CGraph.cpp:



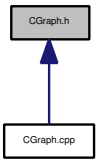
14.7.1 Detailed Description

14.7.2 Variable Documentation

14.7.2.1 IRenderer::Factory registerFactoryG("graph", CGraph::createObj)

14.8 CGraph.h File Reference

Class to render an graph using the root libraries. This graph shows which files directly or indirectly include this file:



Classes

- class **CGraph**

  *Implementation of **IRenderer** (p. 160) interface to render an graph using root libraries.*

14.8.1 Detailed Description

Date:

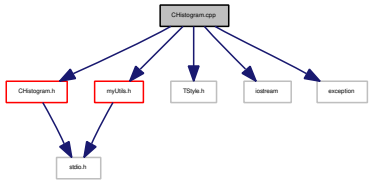
Created on: 01-oct-2009

Author:

Author: Claudio Salvadori

14.9 CHistogram.cpp File Reference

Histogram class method development. Include dependency graph for CHistogram.cpp:



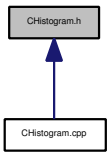
14.9.1 Detailed Description

14.9.2 Variable Documentation

14.9.2.1 IRenderer::Factory registerFactoryH("histogram", CHistogram::createObj)

14.10 CHistogram.h File Reference

Class to render a an histogram using the root libraries. This graph shows which files directly or indirectly include this file:



Classes

- class **CHistogram**

  *Implementation of **IRenderer** (p. 160) interface to render an histogram using root libraries.*

14.10.1 Detailed Description

Date:

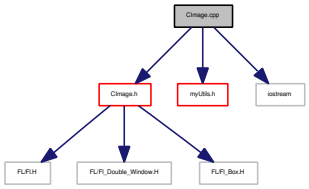
Created on: 01-oct-2009

Author:

Author: Claudio Salvadori

14.11 CImage.cpp File Reference

Image rendering class method development. Include dependency graph for CImage.cpp:



14.11.1 Detailed Description

14.11.2 Define Documentation

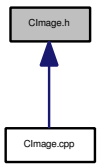
14.11.2.1 #define EXTRA

14.11.3 Variable Documentation

14.11.3.1 IRenderer::Factory registerFactoryI("image", CImage::createObj)

14.12 CImage.h File Reference

Class to render a raw image using FLTK libraries. This graph shows which files directly or indirectly include this file:



Classes

- class **CImage**

  *Implementation of **IRenderer** (p. 160) interface to render a raw image using FLTK libraries.*

14.12.1 Detailed Description

Date:

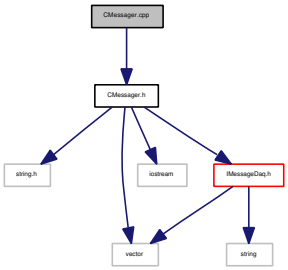
Created on: 01-oct-2009

Author:

Author: Claudio Salvadori

14.13 CMessager.cpp File Reference

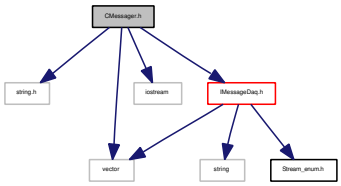
Divisor in messages functions development. Include dependency graph for CMessager.cpp:



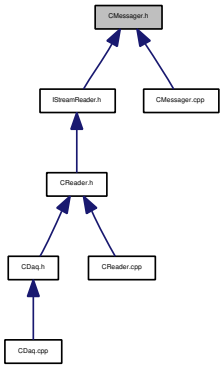
14.13.1 Detailed Description

14.14 CMessager.h File Reference

Implementation of **IMessageDaq** (p. 158) 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. 158) interface to devide in messages a packet.*

14.14.1 Detailed Description

Date:

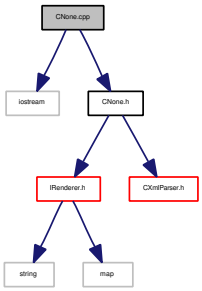
Created on: 26-apr-2009

Author:

Author: Claudio Salvadori

14.15 CNone.cpp File Reference

Registering "none renderer" in the object factory. Include dependency graph for CNone.cpp:



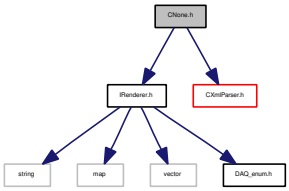
14.15.1 Detailed Description

14.15.2 Variable Documentation

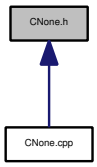
14.15.2.1 IRenderer::Factory registerFactoryN("nothing", CNone::createObj)

14.16 CNone.h File Reference

Class to handle data with no semantic. Include dependency graph for CNone.h:



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



Classes

- class **CNone**

  *Implementation of **IRenderer** (p. 160) interface to handle data with no semantic. This class is implemented to discrard not important character in the stream (like separator).*

14.16.1 Detailed Description

Date:

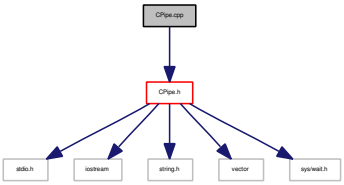
Created on: 23-oct-2009

Author:

Author: Claudio Salvadori

14.17 CPipe.cpp File Reference

Pipe access functions develpoment. Include dependency graph for CPipe.cpp:



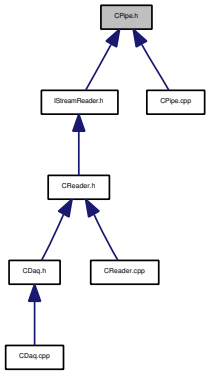
14.17.1 Detailed Description

14.17.2 Define Documentation

14.17.2.1 #define MAX_ACK_LENGHT

14.18 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. 171) 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. 171) structure directives. Daq is client of the pipe created by the server thread.*

14.18.1 Detailed Description

Date:

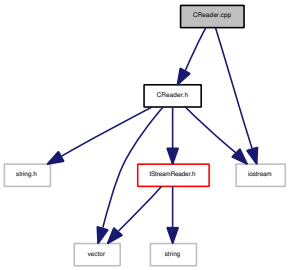
Created on: 22-apr-2009

Author:

Author: Claudio Salvadori

14.19 CReader.cpp File Reference

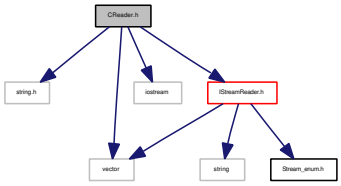
Reader functions development. Include dependency graph for CReader.cpp:



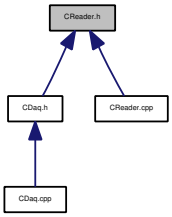
14.19.1 Detailed Description

14.20 CReader.h File Reference

Implemenation for **IStreamReader** (p. 165) interface to read, decode and divide packet to be passed to formatter. This class uses instantiation of the classes **IStreamDaq** (p. 163), **IDecoderDaq** (p. 154), **IMessageDaq** (p. 158). Include dependency graph for CReader.h:



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



Classes

- class **CReader**

  *Implemenation for **IStreamReader** (p. 165) interface to read, decode and divide packet to be passed to formatter. This class uses instantiation of the classes **IStreamDaq** (p. 163), **IDecoderDaq** (p. 154), **IMessageDaq** (p. 158).*

14.20.1 Detailed Description

Date:

Created on: 29-apr-2009

Author:

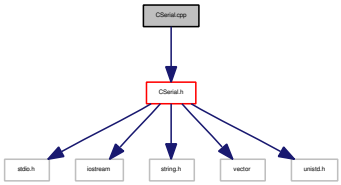
Author: Claudio Salvadori

14.20.2 Define Documentation

14.20.2.1 #define CMESSAGE_H

14.21 CSerial.cpp File Reference

Serial access functions development. Include dependency graph for CSerial.cpp:



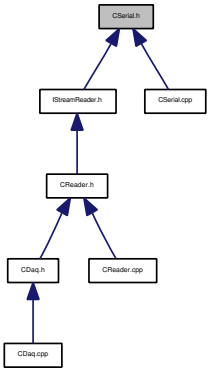
14.21.1 Detailed Description

14.21.2 Define Documentation

14.21.2.1 #define MAX_ACK_LENGHT

14.22 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. 171) 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. 171) structure directives.*

14.22.1 Detailed Description

Date:

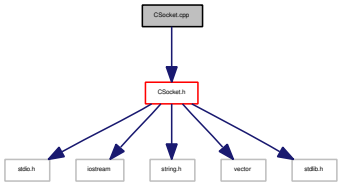
Created on: 27-mar-2009

Author:

Author: Claudio Salvadori

14.23 CSocket.cpp File Reference

Socket access functions develpoment. Include dependency graph for CSocket.cpp:



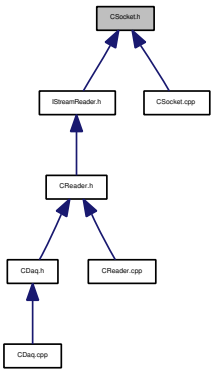
14.23.1 Detailed Description

14.23.2 Define Documentation

14.23.2.1 #define MAX_ACK_LENGHT

14.24 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. 171) 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. 171) structure directives.*

14.24.1 Detailed Description

Date:

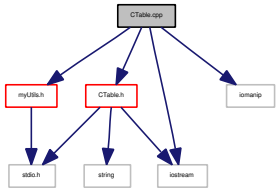
Created on: 22-apr-2009

Author:

Author: Claudio Salvadori

14.25 CTable.cpp File Reference

CTable (p. 117) class method development. Include dependency graph for CTable.cpp:



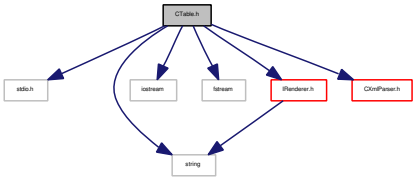
14.25.1 Detailed Description

14.25.2 Variable Documentation

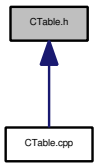
14.25.2.1 IRenderer::Factory registerFactoryT("table", CTable::createObj)

14.26 CTable.h File Reference

Class to write a table inside a file. Include dependency graph for CTable.h:



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



Classes

- class **CTable**

  *Implementation of **IRenderer** (p. 160) interface to write a table inside a file.*

14.26.1 Detailed Description

Date:

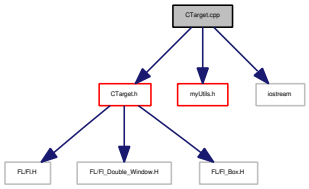
Created on: 01-oct-2009

Author:

Author: Claudio Salvadori

14.27 CTarget.cpp File Reference

Target rendering class method development. Include dependency graph for CTarget.cpp:



14.27.1 Detailed Description

14.27.2 Define Documentation

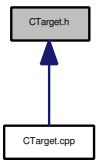
14.27.2.1 #define EXTRA

14.27.3 Variable Documentation

14.27.3.1 IRenderer::Factory registerFactoryTgt("target", CTarget::createObj)

14.28 CTarget.h File Reference

Class to render a black screen to show the position of a target. This graph shows which files directly or indirectly include this file:



Classes

- class **CTarget**

  *Implementation of **IRenderer** (p. 160) interface to render a black screen to show the position of a target using FLTK libraries.*

14.28.1 Detailed Description

Date:

Created on: 31-oct-2009

Author:

Author: Claudio Salvadori

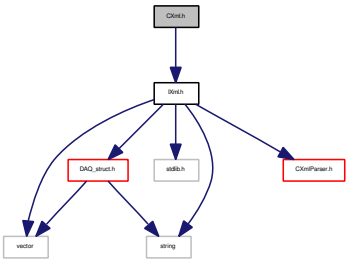
14.29 CXml.cpp File Reference

Xml parser.

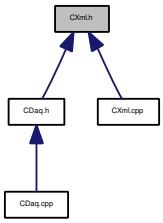
14.29.1 Detailed Description

14.30 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.

14.30.1 Detailed Description

Date:

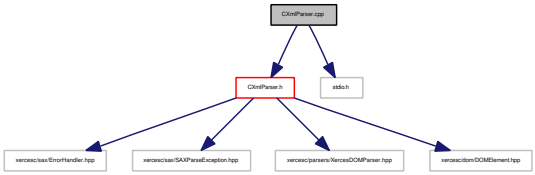
Created on: 30-mar-2009

Author:

Author: Claudio Salvadori

14.31 CXmlParser.cpp File Reference

XML parsing and analysis development functions. Include dependency graph for CXmlParser.cpp:



14.31.1 Detailed Description

14.32 CXmlParser.h File Reference

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

Classes

- class **CXMLParser**

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

14.32.1 Detailed Description

Date:

Created on: 23-mar-2009

Author:

Author: Claudio Salvadori

14.32.2 Define Documentation

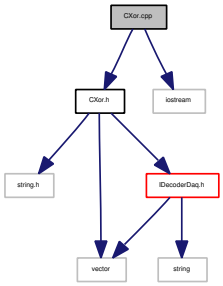
14.32.2.1 #define strX(s)

14.32.3 Variable Documentation

14.32.3.1 XERCES_CPP_NAMESPACE_USE

14.33 CXor.cpp File Reference

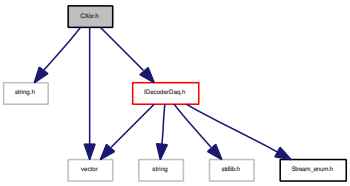
CXor (p. 138) decoder development functions. Include dependency graph for CXor.cpp:



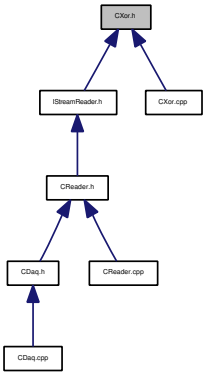
14.33.1 Detailed Description

14.34 CXor.h File Reference

Implementation of **IDecoderDaq** (p. 154) 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. 154) interface to decode the escape character with xor function.*

14.34.1 Detailed Description

Date:

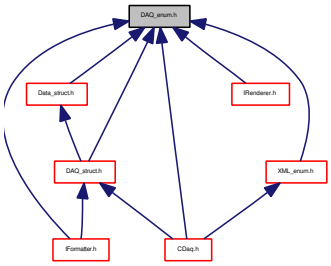
Created on: 26-apr-2009

Author:

Author: Claudio Salvadori

14.35 DAQ_enum.h File Reference

Enumerations for the UI **CDaq** (p. 59) 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**

14.35.1 Detailed Description

Date:

Created on: 1-mag-2009

Author:

Author: Claudio Salvadori

14.35.2 Define Documentation

14.35.2.1 #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

14.35.2.2 #define INT16_S

T_INT16 and T_UINT16 size in number of byte

14.35.2.3 #define INT32_S

T_INT32 and T_UINT32 size in number of byte

14.35.2.4 #define INT64_S

T_INT64 and T_UINT64 size in number of byte

14.35.2.5 #define FLOAT32_S

T_FLOAT32 size in number of byte

14.35.2.6 #define FLOAT64_S

T_FLOAT64 size in number of byte

14.35.3 Enumeration Type Documentation

14.35.3.1 enum decoder_type

Enumeration of available decoder for packet.

Enumerator:

D_NONE No decoder
D_XOR Decoder Xor

14.35.3.2 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)

14.35.3.3 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

14.35.3.4 enum endianess_type

Enumeration of possible endianess defined in the xml file.

Enumerator:

E_BIG Big endian
E_LITTLE Little endian

14.35.3.5 enum handshake_style

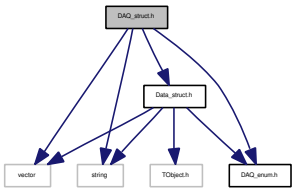
Enumeration of possible type of handshake in the xml file.

Enumerator:

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

14.36 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 **SFile**

  Structure of File.

- 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.

14.36.1 Detailed Description

Date:

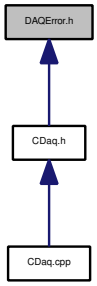
Created on: 10-mar-2009

Author:

Author: Claudio Salvadori

14.37 DAQError.h File Reference

Error enumerations for the UI **CDaq** (p. 59). This graph shows which files directly or indirectly include this file:



14.37.1 Detailed Description

Date:

Created on: 21-08-2009

Author:

Author: Claudio Salvadori

14.37.2 Enumeration Type Documentation

14.37.2.1 enum daq_ret

Enumeration of possible result **CDaq** (p. 59) class method.

Enumerator:

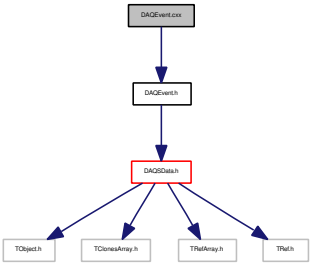
D_RET_OK Ok
D_INIT_ERR Error in parser init
D_PARSE_ERR Error in parsing
D_XML_PARSER_ERR Error in parsing
D_XML_P_FILE_ERR Syntax error in file.xml
D_XML_P_DOM_ERR Error in DOM parsing
D_XML_P_UN_ERR Unexpected exception during parsing
D_P_STYLE_ERR Style error
D_P_TITLE_ERR Title error
D_P_STREAM_ERR 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_P_H_LENGTH_ERR File: header length error
D_P_N_ENTRIES_S_ERR File: number of entries start byte error
D_P_N_ENTRIES_L_ERR File: number of entries length error
D_P_T_LENGTH_ERR File: trailer length error
D_P_ENDIANESS_ERR Endianess 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

14.38 DAQEvent.cxx File Reference

DAQEvent (p. 141) functions development. Include dependency graph for DAQEvent.cxx:



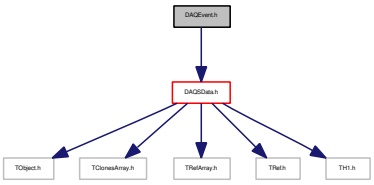
14.38.1 Detailed Description

Author:

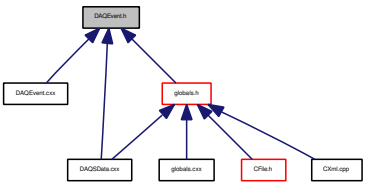
Paolo Pagano

14.39 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.

14.39.1 Detailed Description

Author:

Paolo Pagano

Date:

Wed Jul 8 11:06:26 2009

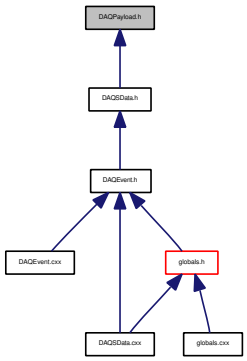
14.40 DAQEventLinkDef.h File Reference

Compiling directive to generate the dictionary.

14.40.1 Detailed Description

14.41 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.

14.41.1 Detailed Description

Author:

Paolo Pagano and Claudio Salvadori

Date:

22-08-2009

14.42 DAQSData.cxx File Reference

DAQSData (p. 146) functions development.

14.42.1 Detailed Description

Author:

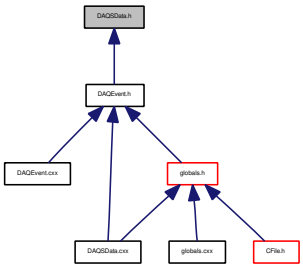
Paolo Pagano

14.42.2 Function Documentation

14.42.2.1 ClassImp (DAQSData)

14.43 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.

14.43.1 Detailed Description

Author:

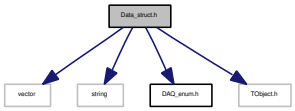
Paolo Pagano and Claudio Salvadori

Date:

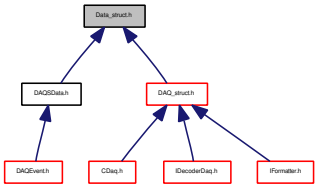
22-08-2009

14.44 Data_struct.h File Reference

Structure of formatted data: data formatted are stored inside **SData** (p. 169) 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.

14.44.1 Detailed Description

Date:

Created on: 05-apr-2009

Author:

Author: Claudio Salvadori

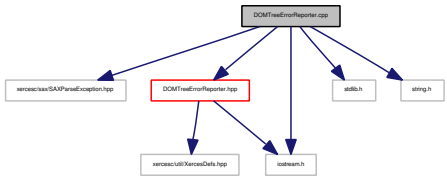
14.45 documentation.h File Reference

Documentation pages.

14.45.1 Detailed Description

14.46 DOMTreeErrorReporter.cpp File Reference

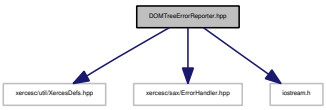
DOMTreeErrorReporter (p. 148) functions development. Include dependency graph for DOMTreeErrorReporter.cpp:



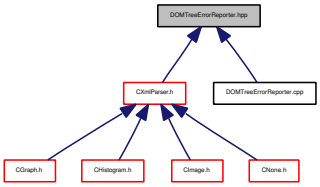
14.46.1 Detailed Description

14.47 DOMTreeErrorReporter.hpp File Reference

Development of the classes **DOMTreeErrorReporter** (p. 148) and **StrX** (p. 185). 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. 151) 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.

14.47.1 Detailed Description

Date:

Created on:2006-11-06

Author:

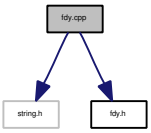
Author: Apache Software Foundation (ASF), documented by Claudio Salvadori

14.47.2 Function Documentation

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

14.48 fdy.cpp File Reference

Vision algorithms development functions. Include dependency graph for fdy.cpp:



Functions

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

14.48.1 Detailed Description

14.48.2 Define Documentation

14.48.2.1 #define GRAY_LEVEL

14.48.3 Function Documentation

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

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

14.48.3.3 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

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

14.48.4 Variable Documentation

14.48.4.1 unsigned int h_old[GRAY_LEVEL]

14.48.4.2 unsigned int h_act[GRAY_LEVEL]

14.48.4.3 unsigned long sum

14.49 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)

14.49.1 Detailed Description

Date:

Created on: 30-apr-2009

Author:

Author: Claudio Salvadori

14.49.2 Function Documentation

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

Parameters:

image Pointer to the actual image

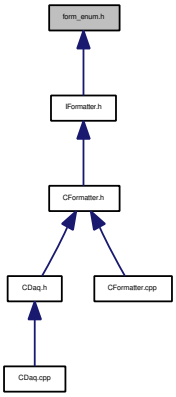
size Image size in bytes

Returns:

Returns the percentage of similarity

14.50 form_enum.h File Reference

Header to enumerate the return value of **CFormatter** (p. 68) class. This graph shows which files directly or indirectly include this file:



14.50.1 Detailed Description

Date:

Created on: 22-apr-2009

Author:

Author: Claudio Salvadori

14.50.2 Enumeration Type Documentation

14.50.2.1 enum f_ret

Enumeration of possible result **IFormatter** (p. 156) 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

14.51 globals.cxx File Reference

Globals functions & variables.

14.51.1 Detailed Description

14.51.2 Function Documentation

14.51.2.1 TROOT root ("hello", "Hello World", initfuncs)

14.51.3 Variable Documentation

14.51.3.1 TFile* rootfile_

14.51.3.2 DAQEvent* event

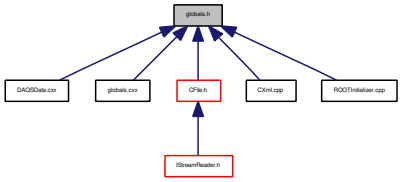
14.51.3.3 TTree* tree

14.51.3.4 VoidFuncPtr_t initfuncs[]

14.51.3.5 unsigned long entries_

14.52 globals.h File Reference

Globals functions & variables. This graph shows which files directly or indirectly include this file:



14.52.1 Detailed Description

14.52.2 Function Documentation

14.52.2.1 void InitGui ()

14.52.3 Variable Documentation

14.52.3.1 TFile* rootfile_

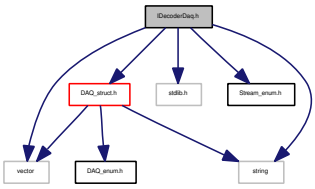
14.52.3.2 DAQEvent* event

14.52.3.3 TTree* tree

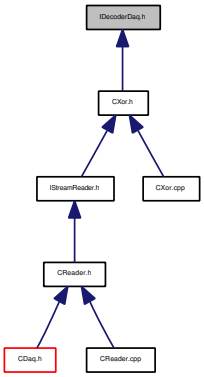
14.52.3.4 unsigned long entries_

14.53 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. 102) 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. 102) class.*

14.53.1 Detailed Description

Date:

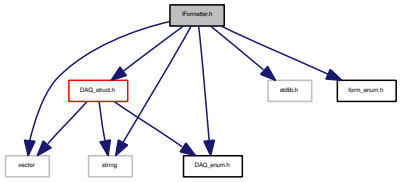
Created on: 22-apr-2009

Author:

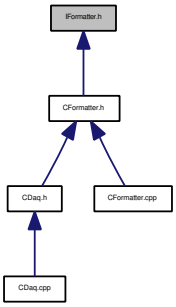
Author: Claudio Salvadori

14.54 IFormatter.h File Reference

Interface for data formatter. Include dependency graph for IFormatter.h:



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



Classes

- class **IFormatter**

  Interface for data formatter.

14.54.1 Detailed Description

Date:

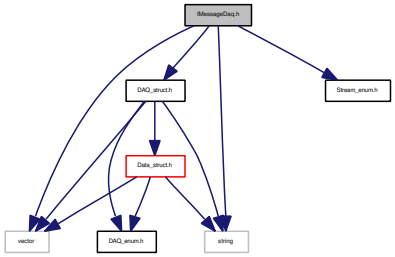
Created on: 10-mar-2009

Author:

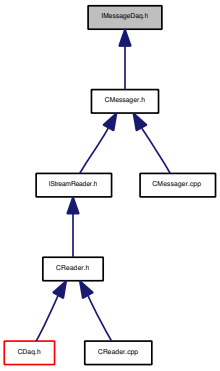
Author: Claudio Salvadori

14.55 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. 102) 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. 102) class.*

14.55.1 Detailed Description

Date:

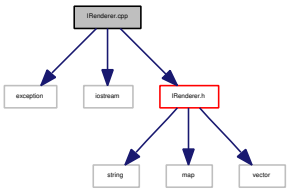
Created on: 25-apr-2009

Author:

Author: Claudio Salvadori

14.56 IRenderer.cpp File Reference

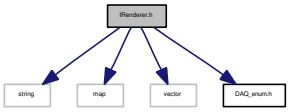
IRenderer (p. 160) class not-virtual method development. Include dependency graph for IRenderer.cpp:



14.56.1 Detailed Description

14.57 IRenderer.h File Reference

Interface to render data (images, histograms, graphs, tables). Include dependency graph for IRenderer.h:



Classes

- class **IRenderer**

  Interface to render data (images, histograms, graphs, tables).

- class **IRenderer::Factory**

  IRenderer *(*p. *160) object factory to handle the renderer.*

14.57.1 Detailed Description

Date:

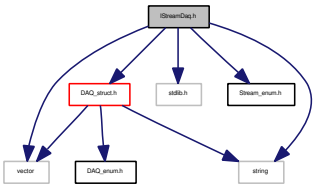
Created on: 05-oct-2009

Author:

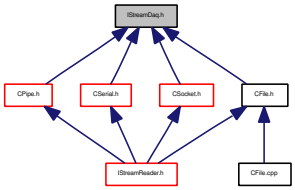
Author: Claudio Salvadori

14.58 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. 102) 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. 102) class.*

14.58.1 Detailed Description

Date:

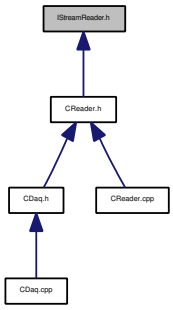
Created on: 22-apr-2009

Author:

Author: Claudio Salvadori

14.59 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.

14.59.1 Detailed Description

Date:

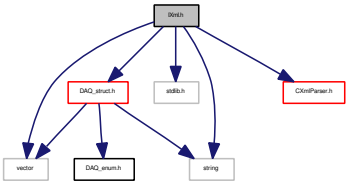
Created on: 29-apr-2009

Author:

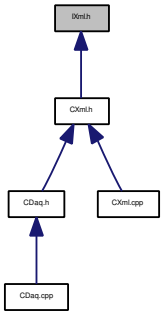
Author: Claudio Salvadori

14.60 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.

14.60.1 Detailed Description

Date:

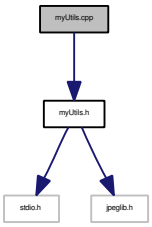
Created on: 29-mar-2009

Author:

Author: Claudio Salvadori

14.61 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)

14.61.1 Detailed Description

14.61.2 Typedef Documentation

14.61.2.1 typedef myJPEG_source_mgr* myJPEG_src_ptr

14.61.3 Function Documentation

14.61.3.1 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

14.61.3.2 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*)

14.61.3.3 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

14.61.3.4 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

14.61.3.5 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

14.61.3.6 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

14.61.3.7 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

14.61.3.8 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

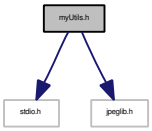
height Image height

14.61.3.9 METHODDEF (void)

14.61.3.10 METHODDEF (boolean)

14.62 myUtils.h File Reference

Conversion functions. This is library usefull for image compression and image conversion. Include dependency graph for myUtils.h:



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)

14.62.1 Detailed Description

Author:

Nastasi Christian

Date:

2007-12-17

14.62.2 Enumeration Type Documentation

14.62.2.1 enum My_draw_color

Colors.

Enumerator:

MY_RED Red

MY_GREEN Green

MY_BLUE Blue

14.62.3 Function Documentation

14.62.3.1 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:

14.62.3.2 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

14.62.3.3 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

14.62.3.4 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

14.62.3.5 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

14.62.3.6 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

14.62.3.7 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

14.62.3.8 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

14.62.3.9 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)

14.63 ROOTInitializer.cpp File Reference

Static class to initialize the repository file.

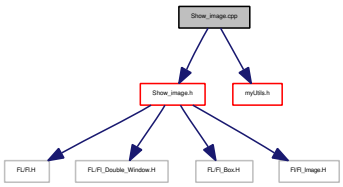
14.63.1 Detailed Description

Author:

Paolo Pagano

14.64 Show_image.cpp File Reference

Image rendering function development. Include dependency graph for Show_image.cpp:



Functions

- void **initRenderer** (unsigned int w, unsigned int h)
- void **renderImage** (unsigned char *image, unsigned int w, unsigned int h, unsigned int channel)
- void **initRenderer_black** (unsigned int w, unsigned int h)
- void **renderImage_black** (unsigned int w, unsigned int h, unsigned int x, unsigned int y)

14.64.1 Detailed Description

14.64.2 Function Documentation

14.64.2.1 void initRenderer (unsigned int *w*, unsigned int *h*)

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

Todo

Add parameter to dimension the size of the windows

14.64.2.2 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

14.64.2.3 void initRenderer_black (unsigned int *w*, unsigned int *h*)

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

Todo

Add parameter to dimension the size of the windows

14.64.2.4 void renderImage_black (unsigned int *w*, unsigned int *h*, unsigned int *x*, unsigned int *y*)

Function to rendet the a black image.

Parameters:

w image width
h image height
x point x component
y point y component

14.64.2.5 void initRendererSX (unsigned int *w*, unsigned int *h*)

14.64.2.6 void renderImageSX (unsigned char * *image*, unsigned int *w*, unsigned int *h*, unsigned int *channel*)

14.64.2.7 void initRendererDX (unsigned int *w*, unsigned int *h*)

14.64.2.8 void renderImageDX (unsigned char * *image*, unsigned int *w*, unsigned int *h*, unsigned int *channel*)

14.64.2.9 void initRendererDisp (unsigned int *w*, unsigned int *h*)

14.64.2.10 void renderImageDisp (unsigned char * *image*, unsigned int *w*, unsigned int *h*, unsigned int *channel*)

14.64.3 Variable Documentation

14.64.3.1 Fl_Double_Window* screen_wnd_SX

14.64.3.2 Fl_Double_Window* screen_wnd_DX

14.64.3.3 Fl_Double_Window* screen_wnd_Disp

14.64.3.4 Fl_Box* screen_img_SX

14.64.3.5 Fl_Box* screen_img_DX

14.64.3.6 Fl_Box* screen_img_Disp

14.64.3.7 Fl_Double_Window* screen_wnd

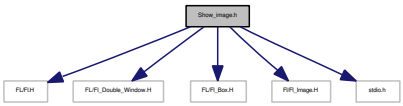
14.64.3.8 Fl_Box* screen_img

14.64.3.9 Fl_Double_Window* screen_wnd_black

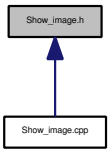
14.64.3.10 Fl_Box* screen_img_black

14.65 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** (unsigned int w, unsigned int h)
- void **renderImage** (unsigned char *image, unsigned int w, unsigned int h, unsigned int channel)
- void **initRenderer_black** (unsigned int w, unsigned int h)
- void **renderImage_black** (unsigned int w, unsigned int h, unsigned int x, unsigned int y)

14.65.1 Detailed Description

Date:

Created on: 01-mag-2009

Author:

Author: Claudio Salvadori

14.65.2 Function Documentation

14.65.2.1 void initRenderer (unsigned int *w*, unsigned int *h*)

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

See also:

renderImage (p. 262)

Parameters:

w image width

h image height

Todo

Add parameter to dimension the size of the windows

14.65.2.2 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

14.65.2.3 void initRenderer_black (unsigned int *w*, unsigned int *h*)

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

See also:

renderImage (p. 262)

Parameters:

w image width

h image height

Todo

Add parameter to dimension the size of the windows

14.65.2.4 void renderImage_black (unsigned int *w*, unsigned int *h*, unsigned int *x*, unsigned int *y*)

Function to rendet the a black image.

Parameters:

w image width

h image height

x point x component

y point y component

14.65.2.5 void initRendererSX (unsigned int *w*, unsigned int *h*)

14.65.2.6 void renderImageSX (unsigned char * *image*, unsigned int *w*, unsigned int *h*, unsigned int *channel*)

14.65.2.7 void initRendererDX (unsigned int *w*, unsigned int *h*)

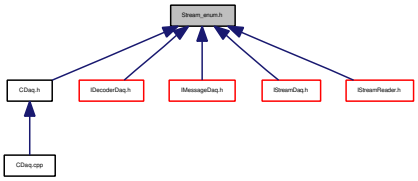
14.65.2.8 void renderImageDX (unsigned char * *image*, unsigned int *w*, unsigned int *h*, unsigned int *channel*)

14.65.2.9 void initRendererDisp (unsigned int *w*, unsigned int *h*)

14.65.2.10 void renderImageDisp (unsigned char * *image*, unsigned int *w*, unsigned int *h*, unsigned int *channel*)

14.66 Stream_enum.h File Reference

All enumeration used by the data reader. This graph shows which files directly or indirectly include this file:



14.66.1 Detailed Description

Date:

Created on: 29-apr-2009

Author:

Author: Claudio Salvadori

14.66.2 Enumeration Type Documentation

14.66.2.1 enum s_ret

Enumeration of possible result **IStreamReader** (p. 165) 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

14.66.2.2 enum s_read_style

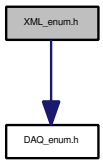
Style of reading.

Enumerator:

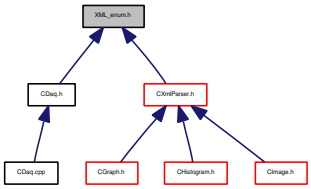
S_STYLE_LENGTH Reader based on the lenght
S_STYLE_S_E Reader based on strart and end character

14.67 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:



14.67.1 Detailed Description

Date:

Created on: 24-apr-2009

Author:

Author: Claudio Salvadori

14.67.2 Enumeration Type Documentation

14.67.2.1 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

14.67.2.2 enum x_ret

Enumeration of possible result **CXml** (p. 125) 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
X_P_H_LENGTH_ERR File: header length error
X_P_N_ENTRIES_S_ERR File: number of entries start byte error
X_P_N_ENTRIES_L_ERR File: number of entries length error
X_P_T_LENGTH_ERR File: trailer length error
X_P_ENDIANESS_ERR Endianess error

14.67.2.3 enum x_msg_type

Enumeration of possible type to write character.

Enumerator:

X_S_T_HEX Hexadecimal
X_S_T_CHAR Character