objectDump

Generated by Doxygen 1.8.5

Wed Jun 10 2015 14:57:38

Contents

Chapter 1

objectDump application

objectDump is an useful support to data acquisition with CAEN digitizers.The application has been tested with a x742 board. objectDump is widely modular so extending it is easy. The user can adapt the application to his purpose.

Author

Daniele Berto

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Class Documentation

5.1 ApplicationSetup Class Reference

The ApplicationSetup class gets the application settings from the main parameters.

`#include <ApplicationSetup.h>`

Public Member Functions

- int ApplicationSetupDataFileModify (const char ∗application_setup_data_file_path_arg)

  The ApplicationSetupDataFileModify method modifies the path of the data file and opens it.
- FILE ∗ ApplicationSetupGetDataFilePunt ()

  The ApplicationSetupGetDataFilePunt method modifies copies in application_setup_data_file_punt_arg the private attrivute application_setup_data_file_punt.
- FILE ∗ ApplicationSetupGetDataFileSizePunt ()

  The ApplicationSetupGetDataFileSziePunt method modifies copies in application_setup_data_file_punt_arg the private attrivute application_setup_data_file_size_punt.
- void ArgumentsParsing ()

  The ArgumentsParsing() method extracts informations from argc and argv.
- void ApplicationSetupSet (int argc, char ∗∗argv)

  The ApplicationSetupSet method picks up settings informations from the main arguments using ArgumentsParsing() method.
- void FetchInputMode (const char ∗application_setup_input_mode)

  The FetchInputMode method picks up the input mode from argv and puts it in input_mode variable.

Static Public Member Functions

- static ApplicationSetup ∗ Instance ()

  Instance() method is used to implement the singleton design pattern: it returns a reference to ApplicationSetup.

Public Attributes

- int channel_visualized

  The channel_visualized variable indicates the channel being visualized.
- ConfObject application_setup_conf_object

  The application_setup_conf_object variable is used for application settings.
- FILE ∗ application_setup_conf_file

  The application_setup_conf_file variable is used to open the configuration file.

- int argc

  The argc variable is used for counting the number of the main arguments.
- int imset

  The imset variable indicates if the object has been set.
- char ∗∗ argv

  The argv variable contains the main arguments.
- const char ∗ application_setup_conf_file_path

  The application_setup_conf_file_path variable contains the path of the configuration file.
- const char ∗ application_setup_log_file_path

  The application_setup_log_file_path variable contains the path of the log file.
- const char ∗ application_setup_data_file_path

  The application_setup_data_file_path variable contains the path of the rawdata file.
- const char ∗ application_setup_data_file_size_path

  The application_setup_data_file_size_path variable contains the path of the file contains the rawdata sizes.
- const char ∗ application_setup_input_mode

  The application_setup_input_mode variable contains the input mode (as it is inserted with -m flag).
- FILE ∗ application_setup_data_file_punt

  The application_setup_data_file_punt variable contains a pointer to the rawdata file.
- FILE ∗ application_setup_data_file_size_punt

  The application_setup_data_file_size_punt variable contains a pointer to the file contains the rawdata sizes.
- int input_mode

  The input_mode variable contains the input mode chosed by the user.

5.1.1 Detailed Description

The ApplicationSetup class gets the application settings from the main parameters.

Il singleton ApplicationSetup permette di salvare e rendere disponibili in tutte le parti del programma le informazioni necessarie per il funzionamento dell'applicazione. Esse sono inserite dall'utente lanciando il programma o inserendo l'apposito input ad applicazione avviata. Dunque, ApplicationSetup ricava le informazioni da argc e argv, cioe' dagli argomenti del main. Occorre che la funzione main chiami il metodo ApplicationSetupSet(argc, argv) per riempire ApplicationSetup. Gli argomenti del main vengono analizzati in modo classico, utilizzando la funzione "getopt".

Author

Daniele Berto

Definition at line 15 of file ApplicationSetup.h.

5.1.2 Member Function Documentation

5.1.2.1 int ApplicationSetup::ApplicationSetupDataFileModify (const char ∗ *application_setup_data_file_path_arg*)

The ApplicationSetupDataFileModify method modifies the path of the data file and opens it.

Definition at line 82 of file ApplicationSetup.cpp.

5.1.2.2 FILE ∗ ApplicationSetup::ApplicationSetupGetDataFilePunt ( )

The ApplicationSetupGetDataFilePunt method modifies copies in application_setup_data_file_punt_arg the private attrivute application_setup_data_file_punt.

Definition at line 108 of file ApplicationSetup.cpp.

5.1.2.3 FILE ∗ ApplicationSetup::ApplicationSetupGetDataFileSizePunt ( )

The ApplicationSetupGetDataFileSziePunt method modifies copies in application_setup_data_file_punt_arg the private attrivute application_setup_data_file_size_punt.

Definition at line 115 of file ApplicationSetup.cpp.

5.1.2.4 void ApplicationSetup::ApplicationSetupSet (int *argc,* char ∗∗ *argv*)

The ApplicationSetupSet method picks up settings informations from the main arguments using ArgumentsParsing() method.

La funzione ApplicationSetupSet salva nelle varibili argc e argv gli argomenti del main.

The caller has to pass argc and argv of the main function.

Parameters

argc	is the number of the main arguments
argv	are the main arguments.

Returns

void

Definition at line 66 of file ApplicationSetup.cpp.

5.1.2.5 void ApplicationSetup::ArgumentsParsing ( )

The ArgumentsParsing() method extracts informations from argc and argv.

Definition at line 155 of file ApplicationSetup.cpp.

5.1.2.6 void ApplicationSetup::FetchInputMode (const char ∗ *application_setup_input_mode*)

The FetchInputMode method picks up the input mode from argv and puts it in input_mode variable.

This method is called by application_setup_set method.

Returns

void

Definition at line 122 of file ApplicationSetup.cpp.

5.1.2.7 ApplicationSetup ∗ ApplicationSetup::Instance ( ) `[static]`

Instance() method is used to implement the singleton design pattern: it returns a reference to ApplicationSetup.

Definition at line 26 of file ApplicationSetup.cpp.

5.1.3 Member Data Documentation

5.1.3.1 FILE∗ ApplicationSetup::application_setup_conf_file

The application_setup_conf_file variable is used to open the configuration file.

Definition at line 48 of file ApplicationSetup.h.

5.1.3.2 const char∗ ApplicationSetup::application_setup_conf_file_path

The application_setup_conf_file_path variable contains the path of the configuration file.

Definition at line 84 of file ApplicationSetup.h.

5.1.3.3 ConfObject ApplicationSetup::application_setup_conf_object

The application_setup_conf_object variable is used for application settings.

Definition at line 43 of file ApplicationSetup.h.

5.1.3.4 const char∗ ApplicationSetup::application_setup_data_file_path

The application_setup_data_file_path variable contains the path of the rawdata file.

Definition at line 94 of file ApplicationSetup.h.

5.1.3.5 FILE∗ ApplicationSetup::application_setup_data_file_punt

The application_setup_data_file_punt variable contains a pointer to the rawdata file.

Definition at line 109 of file ApplicationSetup.h.

5.1.3.6 const char∗ ApplicationSetup::application_setup_data_file_size_path

The application_setup_data_file_size_path variable contains the path of the file contains the rawdata sizes.

Definition at line 99 of file ApplicationSetup.h.

5.1.3.7 FILE∗ ApplicationSetup::application_setup_data_file_size_punt

The application_setup_data_file_size_punt variable contains a pointer to the file contains the rawdata sizes.

Definition at line 114 of file ApplicationSetup.h.

5.1.3.8 const char∗ ApplicationSetup::application_setup_input_mode

The application_setup_input_mode variable contains the input mode (as it is inserted with -m flag).

Definition at line 104 of file ApplicationSetup.h.

5.1.3.9 const char∗ ApplicationSetup::application_setup_log_file_path

The application_setup_log_file_path variable contains the path of the log file.

Definition at line 89 of file ApplicationSetup.h.

5.1.3.10 int ApplicationSetup::argc

The argc variable is used for counting the number of the main arguments.

Definition at line 53 of file ApplicationSetup.h.

5.1.3.11 char∗∗ ApplicationSetup::argv

The argv variable contains the main arguments.

Definition at line 63 of file ApplicationSetup.h.

5.1.3.12 int ApplicationSetup::channel_visualized

The channel_visualized variable indicates the channel being visualized.

The variable is modified by the command "vistart [channelnumber]".

Definition at line 33 of file ApplicationSetup.h.

5.1.3.13 int ApplicationSetup::imset

The imset variable indicates if the object has been set.

Definition at line 58 of file ApplicationSetup.h.

5.1.3.14 int ApplicationSetup::input_mode

The input_mode variable contains the input mode chosed by the user.

Definition at line 119 of file ApplicationSetup.h.

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

- ApplicationSetup.h
- ApplicationSetup.cpp

5.2 ChannelObject Class Reference

The ChannelObject class picks up all the settings taken from the configuration file regarding one channel of the digitizer.

```
#include <ConfObject.h>
```

Public Member Functions

- ChannelObject ()

  The ChannelObject() constructor sets all the values of the channel_object attributes to -1.
- void PrintChannel ()

  The PrintChannel() method prints all the channel_object attributes.

Public Attributes

- int set

  The set variable indicates if a channel has been set or not.
- int numChannel

  The numChannel variable indicates the number of the channel.
- int enable_input

  The enable_input variable indicates if a channel can receive input or not.
- int dc_offset

The variable dc_offset indicates the dc_offset being applicated to the channel.

- int trigger_threshold

  The variable trigger_threshold contains the threshold for the channel auto trigger.

- int channel_trigger

  The variable channel_trigger contains the channel auto trigger settings.

5.2.1 Detailed Description

The ChannelObject class picks up all the settings taken from the configuration file regarding one channel of the digitizer.

The number of the channel is indicated by numChannel variable. If the channel has not been set, the set variable is set to -1 like the other unsetted variables.

Author

Daniele Berto

Definition at line 17 of file ConfObject.h.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 ChannelObject::ChannelObject ( )

The ChannelObject() constructor sets all the values of the channel_object attributes to -1.

Definition at line 56 of file ConfObject.cpp.

5.2.3 Member Function Documentation

5.2.3.1 void ChannelObject::PrintChannel ( )

The PrintChannel() method prints all the channel_object attributes.

Returns

void

Definition at line 88 of file ConfObject.cpp.

5.2.4 Member Data Documentation

5.2.4.1 int ChannelObject::channel_trigger

The variable channel_trigger contains the channel auto trigger settings.

Values: 0 = ACQUISITION_ONLY, 1 = ACQUISITION_AND_TRGOUT, 2 = DISABLED, other=UNSET.

Definition at line 50 of file ConfObject.h.

5.2.4.2 int ChannelObject::dc_offset

The variable dc_offset indicates the dc_offset being applicated to the channel.

Definition at line 39 of file ConfObject.h.

5.2.4.3 int ChannelObject::enable_input

The enable_input variable indicates if a channel can receive input or not.

Definition at line 34 of file ConfObject.h.

5.2.4.4 int ChannelObject::numChannel

The numChannel variable indicates the number of the channel.

Definition at line 29 of file ConfObject.h.

5.2.4.5 int ChannelObject::set

The set variable indicates if a channel has been set or not.

Definition at line 24 of file ConfObject.h.

5.2.4.6 int ChannelObject::trigger_threshold

The variable trigger_threshold contains the threshold for the channel auto trigger.

Definition at line 44 of file ConfObject.h.

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

- ConfObject.h
- ConfObject.cpp

5.3 CommunicationObject Class Reference

```
#include <CommunicationObject.h>
```

Public Member Functions

- void GetCommand (int ∗socketid, int ∗command)

  The GetCommand method copies in the parameters the socket id of the sender and the command sent by it.
- TcpUser GetCommand ()

  The GetCommand method with no arguments return a command sent by someone to the application.
- void CommunicationObjectInit ()

  The CommunicationObjectInit method creates all the things needed by the threads.
- void Main ()

  The main_thread initializes a socket master and listen to connection requests.
- void Worker (void ∗socket_desc)

  The worker_thread manage the incoming input from a client and put it in the circular buffer command[MAXCOMMA-ND].
- void Finish ()

  The Finish method sets to 0 the go variable for calling forth the threads to exit.
- CommunicationObject ()

  The ComunicationObject constructor sets to 0 num_mex and coda variables.

Public Attributes

- int num_mex
  The num_mex variable records the number of messages stored in the circulare buffer command[MAXCOMMAND].
- int testa
  The testa variable is the head pointer.
- int coda
  The coda variable is the tail pointer.
- int go
  The go variable controls the main and the worker threads.
- TcpUser command [3]
  command[MAXCOMMAND] is the circular buffer where the worker threads put the input sent by the clients.
- thread ∗ main_thread
  The main_thread variable is the main thread handler.
- thread ∗ worker_thread
  The thread id of the Worker thread.
- mutex ReservedKeyBoardInputArea
  A mutex used to implement producer-consumer model.
- condition_variable BlockedProducerInput
  A cond variable used to implement producer-consumer model.
- condition_variable BlockedConsumerInput
  A cond variable used to implement producer-consumer model.
- mutex Acquisition_Mutex
  A mutex used to implement producer-consumer model.
- condition_variable Acquisition_Cond1
  A cond variable used to implement producer-consumer model.
- condition_variable Acquisition_Cond2
  A cond variable used to implement producer-consumer model.

5.3.1 Detailed Description

Definition at line 21 of file CommunicationObject.h.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 CommunicationObject::CommunicationObject ( )

The ComunicationObject constructor sets to 0 num_mex and coda variables.

It also sets to 1 testa variable.

Definition at line 203 of file CommunicationObject.cpp.

5.3.3 Member Function Documentation

5.3.3.1 void CommunicationObject::CommunicationObjectInit ( )

The CommunicationObjectInit method creates all the things needed by the threads.

This method initializes the mutexes and the cond variables and sets go to 1 and creates the main_thread.

Returns

void

Definition at line 150 of file CommunicationObject.cpp.

5.3.3.2 void CommunicationObject::Finish ( )

The Finish method sets to 0 the go variable for calling forth the threads to exit.

Returns

void

Definition at line 191 of file CommunicationObject.cpp.

5.3.3.3 void CommunicationObject::GetCommand (int ∗ *socketid,* int ∗ *command*)

The GetCommand method copies in the parameters the socket id of the sender and the command sent by it.

Parameters

socketid	is the socket id of the sender.
command	is the command sent by the sender.

Returns

void

5.3.3.4 TcpUser CommunicationObject::GetCommand ( )

The GetCommand method with no arguments return a command sent by someone to the application.

Returns

The command sent by someone.
TcpUser

Definition at line 160 of file CommunicationObject.cpp.

5.3.3.5 void CommunicationObject::Main ( )

The main_thread initializes a socket master and listen to connection requests.

It creates a worker_thread for each incoming connection. Where variable go is set to zero it closes the socket master and finish.

Definition at line 38 of file CommunicationObject.cpp.

5.3.3.6 void CommunicationObject::Worker (void ∗ *socket_desc*)

The worker_thread manage the incoming input from a client and put it in the circular buffer command[MAXCOMM-AND].

Every worker_thread is created by the thread_main.

Parameters

socket_desc	is the socket id of the client served by the worker_thread

Definition at line 94 of file CommunicationObject.cpp.

5.3.4 Member Data Documentation

5.3.4.1 condition_variable CommunicationObject::Acquisition_Cond1

A cond variable used to implement producer-consumer model.

Definition at line 128 of file CommunicationObject.h.

5.3.4.2 condition_variable CommunicationObject::Acquisition_Cond2

A cond variable used to implement producer-consumer model.

Definition at line 133 of file CommunicationObject.h.

5.3.4.3 mutex CommunicationObject::Acquisition_Mutex

A mutex used to implement producer-consumer model.

Definition at line 123 of file CommunicationObject.h.

5.3.4.4 condition_variable CommunicationObject::BlockedConsumerInput

A cond variable used to implement producer-consumer model.

Definition at line 118 of file CommunicationObject.h.

5.3.4.5 condition_variable CommunicationObject::BlockedProducerInput

A cond variable used to implement producer-consumer model.

Definition at line 113 of file CommunicationObject.h.

5.3.4.6 int CommunicationObject::coda

The coda variable is the tail pointer.

Definition at line 38 of file CommunicationObject.h.

5.3.4.7 TcpUser CommunicationObject::command[3]

command[MAXCOMMAND] is the circular buffer where the worker threads put the input sent by the clients.

When get_command method is called, a value is picked by the circular buffer and returned to the caller. The access to the buffer is managed by a producers-consumer schema: the worker threads are the producers, the get_command method is the consumer.

Definition at line 87 of file CommunicationObject.h.

5.3.4.8 int CommunicationObject::go

The go variable controls the main and the worker threads.

Definition at line 43 of file CommunicationObject.h.

5.3.4.9 thread∗ CommunicationObject::main_thread

The main_thread variable is the main thread handler.

Definition at line 98 of file CommunicationObject.h.

5.3.4.10 int CommunicationObject::num_mex

The num_mex variable records the number of messages stored in the circulare buffer command[MAXCOMMAND].

Definition at line 28 of file CommunicationObject.h.

5.3.4.11 mutex CommunicationObject::ReservedKeyBoardInputArea

A mutex used to implement producer-consumer model.

Definition at line 108 of file CommunicationObject.h.

5.3.4.12 int CommunicationObject::testa

The testa variable is the head pointer.

Definition at line 33 of file CommunicationObject.h.

5.3.4.13 thread∗ CommunicationObject::worker_thread

The thread id of the Worker thread.

Definition at line 103 of file CommunicationObject.h.

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

- CommunicationObject.h
- CommunicationObject.cpp

5.4 ConfigurationConsistence Class Reference

The ConfigurationConsistence class provides useful methods for checking the consistence of a configuration file.

```
#include <ConfigurationConsistence.h>
```

Public Member Functions

- int ConfigurationConsistenceConfFileInit (const char ∗conf_file_path)

  The ConfigurationConsistenceConfFileInit checks if the OPEN attribute in the configuration file is well formed.
- int ConfigurationConsistenceConfFileInitNoPrint (const char ∗conf_file_path)

  The ConfigurationConsistenceConfFileInit checks if the OPEN attribute in the configuration file is well formed.
- int ConfigurationConsistenceConfFileSetupEssentialWithInitCheck (const char ∗conf_file_path)

  The ConfigurationConsistenceConfFileSetupEssentialWithInitCheck checks if the OPEN attribute in the configuration file is well formed and if the other essentials setup attribute are well formed too.
- int ConfigurationConsistenceConfFileSetupEssentialWithInitCheckNoPrint (const char ∗conf_file_path)

  The ConfigurationConsistenceConfFileSetupEssentialWithInitCheck checks if the other essentials setup attribute are well formed too.
- int ConfigurationConsistenceConfFileSetupEssentialWithoutInitCheck (const char ∗conf_file_path)

The ConfigurationConsistenceConfFileSetupEssentialWithInitCheck checks if the other essentials setup attribute are well formed.

- int ConfigurationConsistenceConfFileSetupEssentialWithoutInitCheckNoPrint (const char ∗conf_file_path)

  The ConfigurationConsistenceConfFileSetupEssentialWithInitCheck checks if the other essentials setup attribute are well formed.

5.4.1 Detailed Description

The ConfigurationConsistence class provides useful methods for checking the consistence of a configuration file.

This class require Analizzatore.h parser.

Author

Daniele Berto

Definition at line 12 of file ConfigurationConsistence.h.

5.4.2 Member Function Documentation

5.4.2.1 int ConfigurationConsistence::ConfigurationConsistenceConfFileInit (const char ∗ *conf_file_path*)

The ConfigurationConsistenceConfFileInit checks if the OPEN attribute in the configuration file is well formed.

It prints recognized line.

Parameters

conf_file_path	is the path of the configuration file

Returns

int 0 if no errors are detected.

Definition at line 21 of file ConfigurationConsistence.cpp.

5.4.2.2 int ConfigurationConsistence::ConfigurationConsistenceConfFileInitNoPrint (const char ∗ *conf_file_path*)

The ConfigurationConsistenceConfFileInit checks if the OPEN attribute in the configuration file is well formed.

It does not print recognized line.

Parameters

conf_file_path	is the path of the configuration file

Returns

int 0 if no errors are detected.

Definition at line 93 of file ConfigurationConsistence.cpp.

5.4.2.3 int ConfigurationConsistence::ConfigurationConsistenceConfFileSetupEssentialWithInitCheck (const char ∗ *conf_file_path*)

The ConfigurationConsistenceConfFileSetupEssentialWithInitCheck checks if the OPEN attribute in the configuration file is well formed and if the other essentials setup attribute are well formed too.

It prints recognized line.

Parameters

conf_file_path	is the path of the configuration file

Returns

int 0 if no errors are detected.

Definition at line 147 of file ConfigurationConsistence.cpp.

5.4.2.4 int ConfigurationConsistence::ConfigurationConsistenceConfFileSetupEssentialWithInitCheckNoPrint (const char ∗ *conf_file_path*)

The ConfigurationConsistenceConfFileSetupEssentialWithInitCheck checks if the other essentials setup attribute are well formed too.

It does not print recognized line.

Parameters

conf_file_path	is the path of the configuration file

Returns

int 0 if no errors are detected.

Definition at line 184 of file ConfigurationConsistence.cpp.

5.4.2.5 int ConfigurationConsistence::ConfigurationConsistenceConfFileSetupEssentialWithoutInitCheck (const char ∗ *conf_file_path*)

The ConfigurationConsistenceConfFileSetupEssentialWithInitCheck checks if the other essentials setup attribute are well formed.

It prints recognized line.

Parameters

conf_file_path	is the path of the configuration file

Returns

int 0 if no errors are detected.

Definition at line 220 of file ConfigurationConsistence.cpp.

5.4.2.6 int ConfigurationConsistence::ConfigurationConsistenceConfFileSetupEssentialWithoutInitCheckNoPrint (const char ∗ *conf_file_path*)

The ConfigurationConsistenceConfFileSetupEssentialWithInitCheck checks if the other essentials setup attribute are well formed.

It does not print recognized line.

Parameters

conf_file_path	is the path of the configuration file

Returns

int 0 if no errors are detected.

Definition at line 254 of file ConfigurationConsistence.cpp.

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

- ConfigurationConsistence.h
- ConfigurationConsistence.cpp

5.5 ConfObject Class Reference

The ConfObject class picks up all the settings taken from the configuration file.

`#include <ConfObject.h>`

Public Member Functions

- ConfObject ()

  The ConfObject constructor sets all the values of the conf_object attributes to -1.
- void PrintAll ()

  The PrintAll () method prints all the conf_object attributes.
- void PrintAllHuman ()

  The PrintAllHuman () method prints all the conf_object attributes and interprets their means.

Public Attributes

- int LinkType

  The LinkType variable indicates the physical communication channel.
- int LinkNumber

  In case of USB, the link numbers are assigned by the PC when you connect the cable to the device.
- int ConetNode

  The CONET node identifies which device in the Daisy chain is being addressed.
- int VMEBaseAddress

  The VME Base Address of the board (rotary switches setting) expressed as a 32-bit number.
- int DSR4_Frequency

  The DSR4_Frequency indicates the sampling frequency (for X742 boards only).
- int output_file_format

  The output_file_format variable indicates the format of the output file.
- char ∗ gnuplot

  The gnuplot variable indicates the path to the gnuplot program.
- int header_yes_no

  The header_yes_no variable indicates if the output file has to have a header.
- int record_length

  The record_length variable indicates the size of the acquisition window, that is the number of samples that belong to it.
- int test_pattern

  The test_pattern variable indicates if the board has to produce a test wave.
- int desmod

The desmod variable indicates if the board has to enable the double edge sampling mode (only for the models 731 and 751).

- int external_trigger_acquisition_mode

  The external_trigger_acquisition_mode variable indicates the operating mode of the external trigger.
- int fast_trigger_acquisition_mode

  The fast_trigger_acquisition_mode variable indicates the operating mode of the TRn.
- int enable_fast_trigger_digitizing

  The enable_fast_trigger_digitizing variable (only x742 model) enables/disables the presence of the TRn signal in the data readout.
- int max_num_events_BLT

  The max_num_events_BLT variable indicates the maximum number of events to read out in one Block Transfer.
- int decimation_factor

  The decimation factor variable (only for 740 model) changes the decimation factor for the acquisition.
- int post_trigger

  post trigger size in percent of the whole acquisition window.
- int rising_falling

  rising_falling variable decides whether the trigger occurs on the rising or falling edge of the signal.
- int use_interrupt

  The variable use_interrupt represents the number of events that must be ready for the readout when the IRQ is asserted.
- int nim_ttl

  The variable nim_ttl indicates the type of the front panel I/O LEMO connectors.
- int Address_register

  The variable Address_register contains the address of the register which has to be written at the end of the application setup.
- int Mask_register

  The variable Mask_register contains the bitmask to be used for data masking.
- int Data_register

  The variable Data_register represents the value being written.
- int enable_input

  The variable enable_input indicates if the channels can receive input or not.
- int dc_offset

  The variable dc_offset indicates the dc_offset being applicated to all channels.
- int trigger_threshold

  The variable trigger_threshold contains the threshold for the channel auto trigger.
- ChannelObject channels [64]

  The channels[MAXCHANNELOBJECT] array contains the information settings for each channel.
- GroupObject groups [8]

  The groups[MAXGROUPOBJECT] array contains the information settings for each group.
- FastObject fasts [2]

  The fasts[MAXFASTOBJECT] array contains the information settings for each TRn.
- int channel_enable_mask

  The channel_enable_mask variable contains the mask for enabling the input from the channels.
- int group_enable_mask

  The group_enable_mask variable contains the mask for enabling the input from the groups.
- int self_trigger_enable_mask

  The trigger_enable_mask variable contains the mask for enabling the self trigger of the channel or of the groups (it depends on the model family).
- int self_trigger_enable_mask_mode

  The trigger_enable_mask variable contains the mode for enabling the self trigger of the channel or of the groups (it depends on the model family).

5.5.1 Detailed Description

The ConfObject class picks up all the settings taken from the configuration file.

A ConfObject is required by AnalizzaInit and AnalizzaSetup functions (see analizzatore.h file). This class is useful to sum all the setting informations in only one place.

Author

Daniele Berto

Definition at line 174 of file ConfObject.h.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 ConfObject::ConfObject ( )

The ConfObject constructor sets all the values of the conf_object attributes to -1.

Definition at line 14 of file ConfObject.cpp.

5.5.3 Member Function Documentation

5.5.3.1 void ConfObject::PrintAll ( )

The PrintAll () method prints all the conf_object attributes.

Returns

void

Definition at line 337 of file ConfObject.cpp.

5.5.3.2 void ConfObject::PrintAllHuman ( )

The PrintAllHuman () method prints all the conf_object attributes and interprets their means.

Ex: 0=NO, 1=YES, other=UNSET.

Returns

void

Definition at line 141 of file ConfObject.cpp.

5.5.4 Member Data Documentation

5.5.4.1 int ConfObject::Address_register

The variable Address_register contains the address of the register which has to be written at the end of the application setup.

Definition at line 297 of file ConfObject.h.

5.5.4.2 int ConfObject::channel_enable_mask

The channel_enable_mask variable contains the mask for enabling the input from the channels.

Definition at line 342 of file ConfObject.h.

5.5.4.3 ChannelObject ConfObject::channels[64]

The channels[MAXCHANNELOBJECT] array contains the information settings for each channel.

Definition at line 327 of file ConfObject.h.

5.5.4.4 int ConfObject::ConetNode

The CONET node identifies which device in the Daisy chain is being addressed.

In case of USB, ConetNode must be 0.

Definition at line 193 of file ConfObject.h.

5.5.4.5 int ConfObject::Data_register

The variable Data_register represents the value being written.

Definition at line 307 of file ConfObject.h.

5.5.4.6 int ConfObject::dc_offset

The variable dc_offset indicates the dc_offset being applicated to all channels.

Definition at line 317 of file ConfObject.h.

5.5.4.7 int ConfObject::decimation_factor

The decimation factor variable (only for 740 model) changes the decimation factor for the acquisition.

Options are 1 2 4 8 16 32 64 128.

Definition at line 268 of file ConfObject.h.

5.5.4.8 int ConfObject::desmod

The desmod variable indicates if the board has to enable the double edge sampling mode (only for the models 731 and 751).

Values: 0 = NO, 1 = YES, other=UNSET.

Definition at line 239 of file ConfObject.h.

5.5.4.9 int ConfObject::DSR4_Frequency

The DSR4_Frequency indicates the sampling frequency (for X742 boards only).

Values: 0 = 5GHz, 1 = 2.5GHz, 2 = 1GHz.

Definition at line 205 of file ConfObject.h.

5.5.4.10 int ConfObject::enable_fast_trigger_digitizing

The enable_fast_trigger_digitizing variable (only x742 model) enables/disables the presence of the TRn signal in the data readout.

Values: 0 = NO, 1 = YES, other=UNSET.

Definition at line 257 of file ConfObject.h.

5.5.4.11 int ConfObject::enable_input

The variable enable_input indicates if the channels can receive input or not.

Definition at line 312 of file ConfObject.h.

5.5.4.12 int ConfObject::external_trigger_acquisition_mode

The external_trigger_acquisition_mode variable indicates the operating mode of the external trigger.

Values: 0 = ACQUISITION_ONLY, 1 = ACQUISITION_AND_TRGOUT, 2 = DISABLED, other=UNSET.

Definition at line 245 of file ConfObject.h.

5.5.4.13 int ConfObject::fast_trigger_acquisition_mode

The fast_trigger_acquisition_mode variable indicates the operating mode of the TRn.

Values: 0 = ACQUISITION_ONLY, 1 = ACQUISITION_AND_TRGOUT, 2 = DISABLED, other=UNSET.

Definition at line 251 of file ConfObject.h.

5.5.4.14 FastObject ConfObject::fasts[2]

The fasts[MAXFASTOBJECT] array contains the information settings for each TRn.

Definition at line 337 of file ConfObject.h.

5.5.4.15 char∗ ConfObject::gnuplot

The gnuplot variable indicates the path to the gnuplot program.

Definition at line 216 of file ConfObject.h.

5.5.4.16 int ConfObject::group_enable_mask

The group_enable_mask variable contains the mask for enabling the input from the groups.

Definition at line 347 of file ConfObject.h.

5.5.4.17 GroupObject ConfObject::groups[8]

The groups[MAXGROUPOBJECT] array contains the information settings for each group.

Definition at line 332 of file ConfObject.h.

5.5.4.18 int ConfObject::header_yes_no

The header_yes_no variable indicates if the output file has to have a header.

Values: 0 = NO, 1 = YES, other=UNSET.

Definition at line 222 of file ConfObject.h.

5.5.4.19 int ConfObject::LinkNumber

In case of USB, the link numbers are assigned by the PC when you connect the cable to the device.

For other details please consult CAENDigitizer library documentation.

Definition at line 188 of file ConfObject.h.

5.5.4.20 int ConfObject::LinkType

The LinkType variable indicates the physical communication channel.

0 indicates the USB, 1 indicates the Optical Link

Definition at line 182 of file ConfObject.h.

5.5.4.21 int ConfObject::Mask_register

The variable Mask_register contains the bitmask to be used for data masking.

Definition at line 302 of file ConfObject.h.

5.5.4.22 int ConfObject::max_num_events_BLT

The max_num_events_BLT variable indicates the maximum number of events to read out in one Block Transfer.

Definition at line 262 of file ConfObject.h.

5.5.4.23 int ConfObject::nim_ttl

The variable nim_ttl indicates the type of the front panel I/O LEMO connectors.

Values: 1 = TTL, 0 = NIM, other = UNSET

Definition at line 292 of file ConfObject.h.

5.5.4.24 int ConfObject::output_file_format

The output_file_format variable indicates the format of the output file.

Values: 0 = BINARY, 1 = ASCII.

Definition at line 211 of file ConfObject.h.

5.5.4.25 int ConfObject::post_trigger

post trigger size in percent of the whole acquisition window.

Options: 0 to 100. On models 742 there is a delay of about 35nsec on signal Fast Trigger TR; the post trigger is added to this delay.

Definition at line 275 of file ConfObject.h.

5.5.4.26 int ConfObject::record_length

The record_length variable indicates the size of the acquisition window, that is the number of samples that belong to it.

Definition at line 227 of file ConfObject.h.

5.5.4.27 int ConfObject::rising_falling

rising_falling variable decides whether the trigger occurs on the rising or falling edge of the signal.

Values: 1 = FALLING, 0 = RISING, other=UNSET.

Definition at line 281 of file ConfObject.h.

5.5.4.28 int ConfObject::self_trigger_enable_mask

The trigger_enable_mask variable contains the mask for enabling the self trigger of the channel or of the groups (it depends on the model family).

Definition at line 352 of file ConfObject.h.

5.5.4.29 int ConfObject::self_trigger_enable_mask_mode

The trigger_enable_mask variable contains the mode for enabling the self trigger of the channel or of the groups (it depends on the model family).

Definition at line 357 of file ConfObject.h.

5.5.4.30 int ConfObject::test_pattern

The test_pattern variable indicates if the board has to produce a test wave.

Values: 0 = NO, 1 = YES, other=UNSET.

Definition at line 233 of file ConfObject.h.

5.5.4.31 int ConfObject::trigger_threshold

The variable trigger_threshold contains the threshold for the channel auto trigger.

Definition at line 322 of file ConfObject.h.

5.5.4.32 int ConfObject::use_interrupt

The variable use_interrupt represents the number of events that must be ready for the readout when the IRQ is asserted.

Definition at line 286 of file ConfObject.h.

5.5.4.33 int ConfObject::VMEBaseAddress

The VME Base Address of the board (rotary switches setting) expressed as a 32-bit number.

This argument is used only for the VME models accessed through the VME bus and must be 0 in all other cases.

Definition at line 199 of file ConfObject.h.

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

- ConfObject.h
- ConfObject.cpp

5.6 DigitizerErrorObject Class Reference

The DigitizerErrorObject class contains two methods that print the meaning of the CAEN_DGTZ_ErrorCode.

`#include <DigitizerErrorObject.h>`

Public Member Functions

- void DigitizerErrorObjectDebugging (CAEN_DGTZ_ErrorCode ritorno, const char ∗file, const char ∗func, int line)

  The DigitizerErrorObjectDebugging method prints the meaning of the CAEN_DGTZ_ErrorCode.
- void DigitizerErrorObjectPrintError (CAEN_DGTZ_ErrorCode ritorno)

  The DigitizerErrorObjectPrintError method prints the meaning of the CAEN_DGTZ_ErrorCode.
- int DigitizerErrorObjectDebuggingLog (CAEN_DGTZ_ErrorCode ritorno, const char ∗file, const char ∗func, int line, FILE ∗log_file)

  The DigitizerErrorObjectDebuggingLog method prints the meaning of the CAEN_DGTZ_ErrorCode in the logfile.

5.6.1 Detailed Description

The DigitizerErrorObject class contains two methods that print the meaning of the CAEN_DGTZ_ErrorCode.

Author

Daniele Berto

Definition at line 13 of file DigitizerErrorObject.h.

5.6.2 Member Function Documentation

5.6.2.1 void DigitizerErrorObject::DigitizerErrorObjectDebugging (CAEN_DGTZ_ErrorCode *ritorno,* const char ∗ *file,* const char ∗ *func,* int *line*)

The DigitizerErrorObjectDebugging method prints the meaning of the CAEN_DGTZ_ErrorCode.

It is called by other methods using **FILE** preprocessing variable as second paramater, **func** preprocessing variable as third parameter and **LINE** preprocessing variable as fourth parameter.

Parameters

ritorno	is the CAEN_DGTZ_ErrorCode being interpreted.
file	is the file which call the method: it should be the **FILE** preprocessing variable.
func	is the function where the method is called: it should be the **func** preprocessing variable.
line	is the line where the method is called: it should be the **LINE** preprocessing variable.

Returns

void=

Definition at line 15 of file DigitizerErrorObject.cpp.

5.6.2.2 int DigitizerErrorObject::DigitizerErrorObjectDebuggingLog (CAEN_DGTZ_ErrorCode *ritorno,* const char ∗ *file,* const char ∗ *func,* int *line,* FILE ∗ *log_file*)

The DigitizerErrorObjectDebuggingLog method prints the meaning of the CAEN_DGTZ_ErrorCode in the logfile.

Parameters

ritorno	is the CAEN_DGTZ_ErrorCode being interpreted.
file	is the path of the file in which the functions that generated the error code where located.
func	is the name of the function in which the functions that generated the error code where located.
line	is the line number of the functions that generated the error code.
log_file	is the file pointer being written with the interpretation of the CAEN_DGTZ_ErrorCode.

Returns

void

Definition at line 384 of file DigitizerErrorObject.cpp.

5.6.2.3 void DigitizerErrorObject::DigitizerErrorObjectPrintError (CAEN_DGTZ_ErrorCode *ritorno*)

The DigitizerErrorObjectPrintError method prints the meaning of the CAEN_DGTZ_ErrorCode.

Parameters

ritorno	is the CAEN_DGTZ_ErrorCode being interpreted.

Returns

void

Definition at line 221 of file DigitizerErrorObject.cpp.

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

- DigitizerErrorObject.h
- DigitizerErrorObject.cpp

5.7 DigitizerFlowControl Class Reference

The DigitizerFlowControl class controls the flow of execution of the program.

`#include <DigitizerFlowControl.h>`

Public Member Functions

- DigitizerFlowControl ()

  The DigitizerFlowControl constructor sets the digitizer_flow_control_application_setup variable.
- void DigitizerFlowControlStart ()

  The DigitizerFlowControlStart method begins the main control cycle.
- const char ∗ ParseCommand (int recvline)

  The ParseCommand method prints the meaning of the command received via TCP.
- void Help ()

  The help method lists the commands available.
- bool reg_matches (const char ∗str, const char ∗pattern)

  The reg_matcher method compares the string str with the regex pattern.

Public Attributes

- Input mioinput

  The mioinput variable is used to get the input from the stdin.
- CommunicationObject mioTCP

  The mioTCP pointer is used to get the input from the clients via TCP.
- ApplicationSetup ∗ digitizer_flow_control_application_setup

  The digitizer_flow_control_application_setup variable is used to configure the application.

Static Public Attributes

- static pthread_cond_t input_flow_cond

  The input_flow_cond condition variable is used to notify to the object that an input command is ready to be fetched.
- static pthread_mutex_t input_flow_mutex

  The input_flow_mutex is used with the input_flox_cond to guarantee the consistency of the operations.

5.7.1 Detailed Description

The DigitizerFlowControl class controls the flow of execution of the program.

Author

Daniele Berto

Definition at line 12 of file DigitizerFlowControl.h.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 DigitizerFlowControl::DigitizerFlowControl ( )

The DigitizerFlowControl constructor sets the digitizer_flow_control_application_setup variable.

Definition at line 34 of file DigitizerFlowControl.cpp.

5.7.3 Member Function Documentation

5.7.3.1 void DigitizerFlowControl::DigitizerFlowControlStart ( )

The DigitizerFlowControlStart method begins the main control cycle.

Returns

void

Definition at line 43 of file DigitizerFlowControl.cpp.

5.7.3.2 void DigitizerFlowControl::Help ( )

The help method lists the commands available.

Returns

void

Definition at line 798 of file DigitizerFlowControl.cpp.

5.7.3.3 const char ∗ DigitizerFlowControl::ParseCommand (int *recvline*)

The ParseCommand method prints the meaning of the command received via TCP.

Parameters

recvline	is the command to interpret.

Returns

void

Definition at line 831 of file DigitizerFlowControl.cpp.

5.7.3.4 bool DigitizerFlowControl::reg_matches (const char ∗ *str,* const char ∗ *pattern*)

The reg_matcher method compares the string str with the regex pattern.

Returns

bool

Definition at line 777 of file DigitizerFlowControl.cpp.

5.7.4 Member Data Documentation

5.7.4.1 ApplicationSetup∗ DigitizerFlowControl::digitizer_flow_control_application_setup

The digitizer_flow_control_application_setup variable is used to configure the application.

Definition at line 40 of file DigitizerFlowControl.h.

5.7.4.2 pthread_cond_t DigitizerFlowControl::input_flow_cond [static]

The input_flow_cond condition variable is used to notify to the object that an input command is ready to be fetched.

The DigitizerFlowControl object waits on it until an Input object or a CommunicationObject wakes it up with the pthread_cond_signal function.

Definition at line 20 of file DigitizerFlowControl.h.

5.7.4.3 pthread_mutex_t DigitizerFlowControl::input_flow_mutex [static]

The input_flow_mutex is used with the input_flox_cond to guarantee the consistency of the operations.

Definition at line 25 of file DigitizerFlowControl.h.

5.7.4.4 Input DigitizerFlowControl::mioinput

The mioinput variable is used to get the input from the stdin.

Definition at line 30 of file DigitizerFlowControl.h.

5.7.4.5 CommunicationObject DigitizerFlowControl::mioTCP

The mioTCP pointer is used to get the input from the clients via TCP.

Definition at line 35 of file DigitizerFlowControl.h.

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

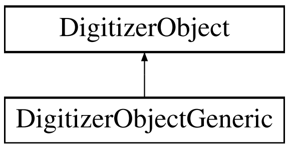
- DigitizerFlowControl.h
- DigitizerFlowControl.cpp

5.8 DigitizerObject Class Reference

The DigitizerObject class envelops CAEN_DGTZ functions from CAENDigitizer library.

```
#include <DigitizerObject.h>
```

Inheritance diagram for DigitizerObject:



Public Member Functions

- DigitizerObject ()

  The DigitizerObject default constructor sets to 0 the variables set_board_info, digitizer_open and set_internal_config.
- DigitizerObject (const char ∗config_file)

  The DigitizerObject (const char ∗) constructor sets to 0 the variables set_board_info, digitizer_open and to 1 set_-internal_config.
- DigitizerObject (ConfObject config)

  The DigitizerObject (ConfObject) constructor sets to 0 the variables set_board_info, digitizer_open and to 1 set_-internal_config.
- void DigitizerObjectSetConfigStructureConfObject (ConfObject config)

  The DigitizerObjectSetConfigStructureConfObject (ConfObject) method copies the ConfObject parameter in the internal_config attribute.
- void DigitizerObjectSetConfigStructureInit (const char ∗config_file)

  The DigitizerObjectSetConfigStructureInit (const char ∗) method scans the file in the path indicates by the parameter using AnalizzaInit and AnalizzaSetup functions from AnalizzatoreLessicale.flex file and stores the retrieved informations in the internal_config attribute.
- void DigitizerObjectSetConfigStructureSetup (const char ∗config_file)

  The DigitizerObjectSetConfigStructureSetup (const char ∗) method scans the file in the path indicates by the parameter using AnalizzaInit and AnalizzaSetup functions from AnalizzatoreLessicale.flex file and stores the retrieved informations in the internal_config attribute.
- int DigitizerObjectOpen ()

  The DigitizerObjectOpen method opens the digitizer using the settings stored in the internal_config attribute.
- int DigitizerObjectClose ()

  The DigitizerObjectOpen method closes the digitizer.
- int DigitizerObjectReset ()

  The DigitizerObjectReset method resets the digitizer.
- int DigitizerObjectGetInfo ()

  The DigitizerObjectGetInfo method gets factory informations from the digitizer and puts them in the BoardInfo attribute.
- CAEN_DGTZ_BoardInfo_t GetBoardInfo ()

  The GetBoardInfo method returns factory informations from the digitizer.
- int GetFamilyCode (int ∗FamilyCode)

  The GetFamilyCode method puts in the parameter FamilyCode the family code of the digitizer.
- int GetFormFactorCode (int ∗FormFactor)

  The GetFormFactorCode method puts in the parameter FormFactor the form factor code of the digitizer.
- void PrintBoardInfo ()

  The PrintBoardInfo method prints to the stdout the factory informations stored in the BoardInfo attribute.
- int DigitizerObjectSetRecordLength (int recordlength)

  The DigitizerObjectSetRecordLength method sets the size of the acquisition window.

- int DigitizerObjectSetGroupEnableMask (int enablemask)

  The DigitizerObjectSetGroupEnableMask method sets the groups enabled to receive input in accordance with the parameter.
- int DigitizerObjectWriteRegister (int registry, int data)

  The DigitizerObjectWriteRegister method writes a digitizer register in accordance with the parameters.
- int DigitizerObjectReadRegister (int registry, int ∗data)

  The DigitizerObjectReadRegister method reads a digitizer register in accordance with the parameters.
- int DigitizerObjectGetDRS4SamplingFrequency (CAEN_DGTZ_DRS4Frequency_t ∗frequenza)

  The DigitizerObjectGetDRS4SamplingFrequency method gets the sampling frequency of the digitizer.
- int DigitizerObjectSetDRS4SamplingFrequency (CAEN_DGTZ_DRS4Frequency_t frequenza)

  The DigitizerObjectSetDRS4SamplingFrequency method sets the sampling frequency of the digitizer.
- int DigitizerObjectSetChannelDCOffset (int channel_mask, int dc_mask)

  The DigitizerObjectSetChannelDCOffset method sets the DCOffset of the channels in accordance with the parameters.
- int DigitizerObjectSetGroupDCOffset (int group_mask, int dc_mask)

  The DigitizerObjectSetGroupDCOffset method sets the DCOffset of the groups in accordance with the parameters.
- int DigitizerObjectSetMaxNumEventsBLT (int MaxNumEventsBLT)

  The DigitizerObjectSetMaxNumEventsBLT method sets the maximum number of events that can be trasferred in a readout cycle.
- int DigitizerObjectSetAcquisitionMode (CAEN_DGTZ_AcqMode_t AcqMode)

  The DigitizerObjectSetAcquisitionMode method sets the data acquisition mode in accordance with the parameter.
- int DigitizerObjectSetExtTriggerInputMode (CAEN_DGTZ_TriggerMode_t TriggerMode)

  The DigitizerObjectSetExtTriggerInputMode method sets how an external trigger has to be used.
- int DigitizerObjectSetSWTriggerMode (CAEN_DGTZ_TriggerMode_t TriggerMode)

  The DigitizerObjectSetSWTriggerMode method sets how a software trigger has to be used.
- int DigitizerObjectSWStartAcquisition ()

  The DigitizerObjectSWStartAcquisition method starts the data acquisition.
- int DigitizerObjectSWStopAcquisition ()

  The DigitizerObjectSWStartAcquisition method stops the data acquisition.

Public Attributes

- CAEN_DGTZ_ErrorCode ret

  The ret variable contains the error code returned by CAEN_DGTZ library functions.
- DigitizerErrorObject ret_error

  The ret_error object is used to print the meaning of CAEN_DGTZ_ErrorCode.
- ConfObject internal_config

  The internal_config object is used to store the setting of the digitizer read from the configuration file.
- int set_board_info

  The set_board_info variable indicates if the board informations have been already picked or not.
- int handle

  The handle variable contains the digitizer handler device.
- CAEN_DGTZ_BoardInfo_t BoardInfo

  The BoardInfo variable contains the informations about the board.
- LogFile ∗ logfile

  The logfile reference is used to write on logfile.

5.8.1 Detailed Description

The DigitizerObject class envelops CAEN_DGTZ functions from CAENDigitizer library.

Author

Daniele Berto

Definition at line 15 of file DigitizerObject.h.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 DigitizerObject::DigitizerObject ( )

The DigitizerObject default constructor sets to 0 the variables set_board_info, digitizer_open and set_internal_-config.

Definition at line 47 of file DigitizerObject.cpp.

5.8.2.2 DigitizerObject::DigitizerObject (const char ∗ *config_file*)

The DigitizerObject (const char ∗) constructor sets to 0 the variables set_board_info, digitizer_open and to 1 set_-internal_config.

The method gets the settings from the configuration file specified in the config_file parameter using AnalizzaInit and AnalizzaSetup functions from AnalizzatoreLessicale.flex file. These settings are copied in the internal_config attribute.

Parameters

config_file	is the path of the configuration file.

Definition at line 18 of file DigitizerObject.cpp.

5.8.2.3 DigitizerObject::DigitizerObject (ConfObject *config*)

The DigitizerObject (ConfObject) constructor sets to 0 the variables set_board_info, digitizer_open and to 1 set_-internal_config.

The method gets the settings from the config parameter and stores them in the internal_config attribute.

Parameters

config	is the ConfObject being copied in the internal_config attribute.

Definition at line 32 of file DigitizerObject.cpp.

5.8.3 Member Function Documentation

5.8.3.1 int DigitizerObject::DigitizerObjectClose ( )

The DigitizerObjectOpen method closes the digitizer.

Returns

int

Definition at line 386 of file DigitizerObject.cpp.

5.8.3.2 int DigitizerObject::DigitizerObjectGetDRS4SamplingFrequency (CAEN_DGTZ_DRS4Frequency_t * *frequenza*)

The DigitizerObjectGetDRS4SamplingFrequency method gets the sampling frequency of the digitizer.

Parameters

frequenza	is where the frequency information will be stored.

Returns

int

Definition at line 314 of file DigitizerObject.cpp.

5.8.3.3 int DigitizerObject::DigitizerObjectGetInfo ( )

The DigitizerObjectGetInfo method gets factory informations from the digitizer and puts them in the BoardInfo attribute.

Returns

int

Definition at line 118 of file DigitizerObject.cpp.

5.8.3.4 int DigitizerObject::DigitizerObjectOpen ( )

The DigitizerObjectOpen method opens the digitizer using the settings stored in the internal_config attribute.

Returns

int

Definition at line 74 of file DigitizerObject.cpp.

5.8.3.5 int DigitizerObject::DigitizerObjectReadRegister (int *registry,* int ∗ *data*)

The DigitizerObjectReadRegister method reads a digitizer register in accordance with the parameters.

Parameters

registry	is the address of the registry being read.
data	are the informations being read from the regitry indicated by registry parameter.

Returns

int

Definition at line 300 of file DigitizerObject.cpp.

5.8.3.6 int DigitizerObject::DigitizerObjectReset ( )

The DigitizerObjectReset method resets the digitizer.

Returns

int

Definition at line 107 of file DigitizerObject.cpp.

5.8.3.7 int DigitizerObject::DigitizerObjectSetAcquisitionMode (CAEN_DGTZ_AcqMode_t *AcqMode*)

The DigitizerObjectSetAcquisitionMode method sets the data acquisition mode in accordance with the parameter.

Parameters

AcqMode	is the data acquisition mode being setted.

Returns

int

Definition at line 373 of file DigitizerObject.cpp.

5.8.3.8 int DigitizerObject::DigitizerObjectSetChannelDCOffset (int *channel_mask,* int *dc_mask*)

The DigitizerObjectSetChannelDCOffset method sets the DCOffset of the channels in accordance with the parameters.

Parameters

channel_mask	indicates the channels being influenced by the DCOffset.
dc_mask	is the DAC value.

Returns

int

Definition at line 339 of file DigitizerObject.cpp.

5.8.3.9 void DigitizerObject::DigitizerObjectSetConfigStructureConfObject (ConfObject *config*)

The DigitizerObjectSetConfigStructureConfObject (ConfObject) method copies the ConfObject parameter in the internal_config attribute.

Parameters

config	is the ConfObject being copied in the internal_config attribute.

Returns

void

Definition at line 41 of file DigitizerObject.cpp.

5.8.3.10 void DigitizerObject::DigitizerObjectSetConfigStructureInit (const char ∗ *config_file*)

The DigitizerObjectSetConfigStructureInit (const char ∗) method scans the file in the path indicates by the parameter using AnalizzaInit and AnalizzaSetup functions from AnalizzatoreLessicale.flex file and stores the retrieved informations in the internal_config attribute.

Parameters

config_file	is the path of the configuration file.

Returns

void

Definition at line 55 of file DigitizerObject.cpp.

5.8.3.11 void DigitizerObject::DigitizerObjectSetConfigStructureSetup (const char ∗ *config_file*)

The DigitizerObjectSetConfigStructureSetup (const char ∗) method scans the file in the path indicates by the parameter using AnalizzaInit and AnalizzaSetup functions from AnalizzatoreLessicale.flex file and stores the retrieved informations in the internal_config attribute.

Parameters

config_file	is the path of the configuration file.

Returns

void

Definition at line 65 of file DigitizerObject.cpp.

5.8.3.12 int DigitizerObject::DigitizerObjectSetDRS4SamplingFrequency (CAEN_DGTZ_DRS4Frequency_t *frequenza*)

The DigitizerObjectSetDRS4SamplingFrequency method sets the sampling frequency of the digitizer.

Parameters

frequenza	is the frequency being setted.

Returns

int

Definition at line 327 of file DigitizerObject.cpp.

5.8.3.13 int DigitizerObject::DigitizerObjectSetExtTriggerInputMode (CAEN_DGTZ_TriggerMode_t *TriggerMode*)

The DigitizerObjectSetExtTriggerInputMode method sets how an external trigger has to be used.

Parameters

TriggerMode	indicates how an external trigger has to be used.

Returns

int

Definition at line 399 of file DigitizerObject.cpp.

5.8.3.14 int DigitizerObject::DigitizerObjectSetGroupDCOffset (int *group_mask,* int *dc_mask*)

The DigitizerObjectSetGroupDCOffset method sets the DCOffset of the groups in accordance with the parameters.

Parameters

group_mask	indicates the groups being influenced by the DCOffset.
dc_mask	is the DAC value.

Returns

int

Definition at line 351 of file DigitizerObject.cpp.

5.8.3.15 int DigitizerObject::DigitizerObjectSetGroupEnableMask (int *enablemask*)

The DigitizerObjectSetGroupEnableMask method sets the groups enabled to receive input in accordance with the parameter.

Parameters

enablemask	is the mask that indicates which groups will receive input.

Returns

int

Definition at line 278 of file DigitizerObject.cpp.

5.8.3.16 int DigitizerObject::DigitizerObjectSetMaxNumEventsBLT (int *MaxNumEventsBLT*)

The DigitizerObjectSetMaxNumEventsBLT method sets the maximum number of events that can be trasferred in a readout cycle.

Parameters

MaxNumEvents-BLT	is the maximum number of events that can be trasferred in a readout cycle.

Returns

int

Definition at line 361 of file DigitizerObject.cpp.

5.8.3.17 int DigitizerObject::DigitizerObjectSetRecordLength (int *recordlength*)

The DigitizerObjectSetRecordLength method sets the size of the acquisition window.

Parameters

recordlength	is the size of the acquisition window.

Returns

int

Definition at line 266 of file DigitizerObject.cpp.

5.8.3.18 int DigitizerObject::DigitizerObjectSetSWTriggerMode (CAEN_DGTZ_TriggerMode_t *TriggerMode*)

The DigitizerObjectSetSWTriggerMode method sets how a software trigger has to be used.

Parameters

TriggerMode	indicates how a software trigger has to be used.

Returns

int

Definition at line 411 of file DigitizerObject.cpp.

5.8.3.19 int DigitizerObject::DigitizerObjectSWStartAcquisition ( )

The DigitizerObjectSWStartAcquisition method starts the data acquisition.

Returns

int

Definition at line 424 of file DigitizerObject.cpp.

5.8.3.20 int DigitizerObject::DigitizerObjectSWStopAcquisition ( )

The DigitizerObjectSWStartAcquisition method stops the data acquisition.

Returns

int

Definition at line 436 of file DigitizerObject.cpp.

5.8.3.21 int DigitizerObject::DigitizerObjectWriteRegister (int *registry*, int *data*)

The DigitizerObjectWriteRegister method writes a digitizer register in accordance with the parameters.

Parameters

registry	is the address of the registry being written.
data	are the informations being written into the regitry indicated by registry parameter.

Returns

int

Definition at line 288 of file DigitizerObject.cpp.

5.8.3.22 CAEN_DGTZ_BoardInfo_t DigitizerObject::GetBoardInfo ( )

The GetBoardInfo method returns factory informations from the digitizer.

Returns

CAEN_DGTZ_BoardInfo_t

5.8.3.23 int DigitizerObject::GetFamilyCode (int ∗ *FamilyCode*)

The GetFamilyCode method puts in the parameter FamilyCode the family code of the digitizer.

Definition at line 131 of file DigitizerObject.cpp.

5.8.3.24 int DigitizerObject::GetFormFactorCode (int ∗ *FormFactor*)

The GetFormFactorCode method puts in the parameter FormFactor the form factor code of the digitizer.

Definition at line 151 of file DigitizerObject.cpp.

5.8.3.25 void DigitizerObject::PrintBoardInfo ( )

The PrintBoardInfo method prints to the stdout the factory informations stored in the BoardInfo attribute.

Returns

void

Definition at line 171 of file DigitizerObject.cpp.

5.8.4 Member Data Documentation

5.8.4.1 CAEN_DGTZ_BoardInfo_t DigitizerObject::BoardInfo

The BoardInfo variable contains the informations about the board.

Definition at line 48 of file DigitizerObject.h.

5.8.4.2 int DigitizerObject::handle

The handle variable contains the digitizer handler device.

Definition at line 43 of file DigitizerObject.h.

5.8.4.3 ConfObject DigitizerObject::internal_config

The internal_config object is used to store the setting of the digitizer read from the configuration file.

Definition at line 33 of file DigitizerObject.h.

5.8.4.4 LogFile* DigitizerObject::logfile

The logfile reference is used to write on logfile.

Definition at line 245 of file DigitizerObject.h.

5.8.4.5 CAEN_DGTZ_ErrorCode DigitizerObject::ret

The ret variable contains the error code returned by CAEN_DGTZ library functions.

Definition at line 23 of file DigitizerObject.h.

5.8.4.6 DigitizerErrorObject DigitizerObject::ret_error

The ret_error object is used to print the meaning of CAEN_DGTZ_ErrorCode.

Definition at line 28 of file DigitizerObject.h.

5.8.4.7 int DigitizerObject::set_board_info

The set_board_info variable indicates if the board informations have been already picked or not.

Definition at line 38 of file DigitizerObject.h.

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

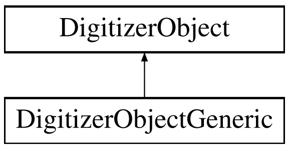
- DigitizerObject.h
- DigitizerObject.cpp

5.9 DigitizerObjectGeneric Class Reference

The DigitizerObjectGeneric class extends the DigitizerObject class whith methods for setting the digitizer using "internal_config" attribute.

```
#include <DigitizerObjectGeneric.h>
```

Inheritance diagram for DigitizerObjectGeneric:



Public Member Functions

- int DigitizerObjectGenericSetRecordLength ()

  The DigitizerObjectGenericSetRecordLength method sets the size of the acquisition window.
- int DigitizerObjectGenericSetMaxNumEventsBLT ()

  The DigitizerObjectGenericSetMaxNumEventsBLT method sets the maximum number of events that can be trasferred in a readout cycle.
- int DigitizerObjectGenericSetEnableMask ()

  The DigitizerObjectGenericSetEnableMask method sets the groups or the channels (it depends on the model family) able to receive input in accordance with the config file settings.
- int DigitizerObjectGenericSetFastTriggerDigitizing ()

  The DigitizerObjectGenericSetFastTriggerDigitizing method sets the TRn input channel (only for the x742 family) in accordance with the config file settings.
- int DigitizerObjectGenericSetSelfTrigger ()

  The DigitizerObjectGenericSetChannelSelfTrigger method sets the groups or the channels (it depends on the model family) self trigger.
- int DigitizerObjectGenericSetChannelSelfTriggerThreshold ()

  The DigitizerObjectGenericSetChannelSelfTriggerThreshold method sets the self trigger threshold of the groups or of the channels (it depends on the model family).
- int DigitizerObjectGenericSetDCOffset ()

  The DigitizerObjectGenericSetEnableMask method sets the DCOffset of the groups or of the channels (it depends on the model family).
- int DigitizerObjectGenericWriteRegister ()

  The DigitizerObjectWriteRegister method writes a digitizer register in accordance with the config file settings.
- int DigitizerObjectGenericReadRegister ()

  The DigitizerObjectGenericReadRegister method reads a digitizer register in accordance with the config file settings.
- int DigitizerObjectGenericGetDRS4SamplingFrequency ()

  The DigitizerObjectGenericGetDRS4SamplingFrequency method gets the sampling frequency of the digitizer.
- int DigitizerObjectGenericSetDRS4SamplingFrequency ()

  The DigitizerObjectGenericSetDRS4SamplingFrequency method sets the sampling frequency of the digitizer.
- int DigitizerObjectGenericSetChannelDCOffset ()

  The DigitizerObjectGenericSetChannelDCOffset method sets the DCOffset of the channels in accordance with the config file settings.
- int DigitizerObjectGenericSetGroupDCOffset ()

  The DigitizerObjectGenericSetGroupDCOffset method sets the DCOffset of the groups in accordance with the config file settings.
- int DigitizerObjectGenericSetTriggerPolarity ()

  The DigitizerObjectGenericSetTriggerPolarity method sets the trigger polarity of a specified channel in accordance with the config file settings.

- int DigitizerObjectGenericSetPostTriggerSize ()

  The DigitizerObjectGenericSetPostTriggerSize method sets the position of the trigger within the acquisition window in accordance with the config file settings.
- int DigitizerObjectGenericSetIOLevel ()

  The DigitizerObjectGenericSetIOLevel method sets the I/0 level in accordance with the config file settings.
- int DigitizerObjectGenericSetAcquisitionMode ()

  The DigitizerObjectGenericSetAcquisitionMode method sets the data acquisition mode in accordance with the config file settings.
- int DigitizerObjectGenericSetExtTriggerInputMode ()

  The DigitizerObjectGenericSetExtTriggerInputMode method sets how an external trigger has to be used.
- int DigitizerObjectGenericSetSWTriggerMode ()

  The DigitizerObjectGenericSetSWTriggerMode method sets how a software trigger has to be used.
- int DigitizerObjectGenericSetDecimationFactor ()

  The DigitizerObjectGenericSetDecimationFactor method sets the decimation factor.
- int DigitizerObjectGenericSetDesMode ()

  The DigitizerObjectGenericSetDesMode method enables the dual edge sampling (DES) mode (for 731 and 751 series only).
- int DigitizerObjectGenericSetTestPattern ()

  The DigitizerObjectGenericSetTestPattern method sets the waveform test bit for debugging.
- int DigitizerObjectGenericSetAllInformations ()

  The DigitizerObjectGenericSetAllInformations method sets the digitizer with the settings marked with "all" in the configuration file.
- DigitizerObjectGeneric ()

  The DigitizerObjectGeneric constructor sets to 0 the set_board_info attribute and gets an instance of the LogFile singleton.

Additional Inherited Members

5.9.1 Detailed Description

The DigitizerObjectGeneric class extends the DigitizerObject class whith methods for setting the digitizer using "internal_config" attribute.

DigitizerObject class provides methods for setting the digitizer specifying the settings in their parameters. The methods of DigitizerObjectGeneric class are parameterless because they use settings provides by internal_config attribute.

Author

Daniele Berto

Definition at line 16 of file DigitizerObjectGeneric.h.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 DigitizerObjectGeneric::DigitizerObjectGeneric ( )

The DigitizerObjectGeneric constructor sets to 0 the set_board_info attribute and gets an instance of the LogFile singleton.

Definition at line 17 of file DigitizerObjectGeneric.cpp.

5.9.3 Member Function Documentation

5.9.3.1 int DigitizerObjectGeneric::DigitizerObjectGenericGetDRS4SamplingFrequency ( )

The DigitizerObjectGenericGetDRS4SamplingFrequency method gets the sampling frequency of the digitizer.

Returns

int

5.9.3.2 int DigitizerObjectGeneric::DigitizerObjectGenericReadRegister ( )

The DigitizerObjectGenericReadRegister method reads a digitizer register in accordance with the config file settings.

Returns

int

5.9.3.3 int DigitizerObjectGeneric::DigitizerObjectGenericSetAcquisitionMode ( )

The DigitizerObjectGenericSetAcquisitionMode method sets the data acquisition mode in accordance with the config file settings.

Returns

int

5.9.3.4 int DigitizerObjectGeneric::DigitizerObjectGenericSetAllInformations ( )

The DigitizerObjectGenericSetAllInformations method sets the digitizer with the settings marked with "all" in the configuration file.

Returns

int

Definition at line 25 of file DigitizerObjectGeneric.cpp.

5.9.3.5 int DigitizerObjectGeneric::DigitizerObjectGenericSetChannelDCOffset ( )

The DigitizerObjectGenericSetChannelDCOffset method sets the DCOffset of the channels in accordance with the config file settings.

Returns

int

5.9.3.6 int DigitizerObjectGeneric::DigitizerObjectGenericSetChannelSelfTriggerThreshold ( )

The DigitizerObjectGenericSetChannelSelfTriggerThreshold method sets the self trigger threshold of the groups or of the channels (it depends on the model family).

Returns

int

Definition at line 447 of file DigitizerObjectGeneric.cpp.

5.9.3.7 int DigitizerObjectGeneric::DigitizerObjectGenericSetDCOffset ( )

The DigitizerObjectGenericSetEnableMask method sets the DCOffset of the groups or of the channels (it depends on the model family).

Returns

int

Definition at line 332 of file DigitizerObjectGeneric.cpp.

5.9.3.8 int DigitizerObjectGeneric::DigitizerObjectGenericSetDecimationFactor ( )

The DigitizerObjectGenericSetDecimationFactor method sets the decimation factor.

This method produces effects only for 740 series.

Returns

int

Definition at line 148 of file DigitizerObjectGeneric.cpp.

5.9.3.9 int DigitizerObjectGeneric::DigitizerObjectGenericSetDesMode ( )

The DigitizerObjectGenericSetDesMode method enables the dual edge sampling (DES) mode (for 731 and 751 series only).

Returns

int

Definition at line 160 of file DigitizerObjectGeneric.cpp.

5.9.3.10 int DigitizerObjectGeneric::DigitizerObjectGenericSetDRS4SamplingFrequency ( )

The DigitizerObjectGenericSetDRS4SamplingFrequency method sets the sampling frequency of the digitizer.

Returns

int

Definition at line 269 of file DigitizerObjectGeneric.cpp.

5.9.3.11 int DigitizerObjectGeneric::DigitizerObjectGenericSetEnableMask ( )

The DigitizerObjectGenericSetEnableMask method sets the groups or the channels (it depends on the model family) able to receive input in accordance with the config file settings.

Returns

int

Definition at line 214 of file DigitizerObjectGeneric.cpp.

5.9.3.12 int DigitizerObjectGeneric::DigitizerObjectGenericSetExtTriggerInputMode ( )

The DigitizerObjectGenericSetExtTriggerInputMode method sets how an external trigger has to be used.

Returns

int

Definition at line 239 of file DigitizerObjectGeneric.cpp.

5.9.3.13 int DigitizerObjectGeneric::DigitizerObjectGenericSetFastTriggerDigitizing ( )

The DigitizerObjectGenericSetFastTriggerDigitizing method sets the TRn input channel (only for the x742 family) in accordance with the config file settings.

Returns

int

Definition at line 487 of file DigitizerObjectGeneric.cpp.

5.9.3.14 int DigitizerObjectGeneric::DigitizerObjectGenericSetGroupDCOffset ( )

The DigitizerObjectGenericSetGroupDCOffset method sets the DCOffset of the groups in accordance with the config file settings.

Returns

int

5.9.3.15 int DigitizerObjectGeneric::DigitizerObjectGenericSetIOLevel ( )

The DigitizerObjectGenericSetIOLevel method sets the I/0 level in accordance with the config file settings.

Returns

int

Definition at line 311 of file DigitizerObjectGeneric.cpp.

5.9.3.16 int DigitizerObjectGeneric::DigitizerObjectGenericSetMaxNumEventsBLT ( )

The DigitizerObjectGenericSetMaxNumEventsBLT method sets the maximum number of events that can be trasferred in a readout cycle.

Returns

int

Definition at line 204 of file DigitizerObjectGeneric.cpp.

5.9.3.17 int DigitizerObjectGeneric::DigitizerObjectGenericSetPostTriggerSize ( )

The DigitizerObjectGenericSetPostTriggerSize method sets the position of the trigger within the acquisition window in accordance with the config file settings.

Returns

int

Definition at line 299 of file DigitizerObjectGeneric.cpp.

5.9.3.18 int DigitizerObjectGeneric::DigitizerObjectGenericSetRecordLength ( )

The DigitizerObjectGenericSetRecordLength method sets the size of the acquisition window.

Returns

int

Definition at line 195 of file DigitizerObjectGeneric.cpp.

5.9.3.19 int DigitizerObjectGeneric::DigitizerObjectGenericSetSelfTrigger ( )

The DigitizerObjectGenericSetChannelSelfTrigger method sets the groups or the channels (it depends on the model family) self trigger.

Returns

int

Definition at line 369 of file DigitizerObjectGeneric.cpp.

5.9.3.20 int DigitizerObjectGeneric::DigitizerObjectGenericSetSWTriggerMode ( )

The DigitizerObjectGenericSetSWTriggerMode method sets how a software trigger has to be used.

Returns

int

5.9.3.21 int DigitizerObjectGeneric::DigitizerObjectGenericSetTestPattern ( )

The DigitizerObjectGenericSetTestPattern method sets the waveform test bit for debugging.

Returns

int

Definition at line 183 of file DigitizerObjectGeneric.cpp.

5.9.3.22 int DigitizerObjectGeneric::DigitizerObjectGenericSetTriggerPolarity ( )

The DigitizerObjectGenericSetTriggerPolarity method sets the trigger polarity of a specified channel in accordance with the config file settings.

Returns

int

5.9.3.23 int DigitizerObjectGeneric::DigitizerObjectGenericWriteRegister ( )

The DigitizerObjectWriteRegister method writes a digitizer register in accordance with the config file settings.

Returns

int

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

- DigitizerObjectGeneric.h
- DigitizerObjectGeneric.cpp

5.10 DigitizerStateMachine Class Reference

```
#include <DigitizerStateMachine.h>
```

Public Member Functions

- void DigitizerStateMachineSetup (const char ∗conf_file)

  The DigitizerStateMachineSetup method sets the digitizer with the settings specified in the configuration file.
- void DigitizerStateMachineStartReading ()

  The DigitizerStateMachineStartReading method begins the data acquisition from the digitizer.
- void DigitizerStateMachinePrintStatus ()

  The DigitizerStateMachinePrintStatus method prints some informations about the threads of the object.
- void DigitizerStateMachineStopReading ()

  The DigitizerStateMachineStopReading method stops the data acquisition from the digitizer.
- void DigitizerStateMachineRawDataInit ()

  The DigitizerStateMachineRawDataInit method creates the Produttore and the Consumatore threads.
- void DigitizerStateMachineQuit ()

  The DigitizerStateMachineQuit method closes the digitizer.
- void DigitizerStateMachineSendSWTrigger ()

  The DigitizerStateMachineSendSWTrigger method sends one software trigger to the digitizer.
- void DigitizerStateStartPreprocessing ()

  The DigitizerStartPreprocessing method starts the preprocessing actions.
- void DigitizerStateStartVisualization ()

  The DigitizerStartVisualization method starts the visualization actions.
- void DigitizerStateStopPreprocessing ()

  The DigitizerStopPreprocessing method starts the preprocessing actions.
- void DigitizerStateStartRawDataWriting ()

  The DigitizerStartRawDataWriting method starts the visualization actions.
- void DigitizerStateStopRawDataWriting ()

  The DigitizerStopRawDataWriting method starts the preprocessing actions.
- void DigitizerStateStopVisualization ()

  The DigitizerStopVisualization method starts the visualization actions.
- void Produttore ()

  The Produttore thread reads data from the digitizer and puts them in the buffers.
- void Consumatore_Dispatcher ()

  The Consumatore thread picks data from the circular_buffer_raw_data[RAWDATAQUEUE] and writes them into the hard disk.
- void Preprocessing ()

The Preprocessing thread picks data from the circular_buffer_preprocessing[PREPROCESSINGQUEUE] and performs the preprocessing data actions.

- void Visualization ()

  The Visualization thread picks data from the circular_buffer_visualization[VISUALIZATIONQUEUE] and visualizes them using the gnuplot program.

- void DigitizerStateMachineInit (const char ∗conf_file)

  The DigitizerStateMachineInit(const char ∗) configures the digitizer.

- DigitizerStateMachine ()

  The DigitizerStateMachine(const char ∗) constructor sets go_general variable to zero.

Public Attributes

- int num_mex_raw_data

  The num_mex_raw_data variable represents the number of messages stored in the circular_buffer_raw_data[RAWDATAQUEUE] attribute.

- int num_mex_preprocessing

  The num_mex_preprocessing variable represents the number of messages stored in the circular_buffer_preprocessing[PREPROCESSINGQUEUE] attribute.

- int num_mex_visualization

  The num_mex_visualization variable represents the number of messages stored in the circular_buffer_visualization[VISUALIZATIONQUEUE] attribute.

- int testa_raw_data

  The testa_raw_data variable is the pointer to the head of the circular_buffer_raw_data[RAWDATAQUEUE] attribute.

- int testa_preprocessing

  The testa_preprocessing variable is the pointer to the head of the circular_buffer_preprocessing[PREPROCESSINGQUEUE] attribute.

- int testa_visualization

  The testa_visualization variable is the pointer to the head of the circular_buffer_visualization[VISUALIZATIONQUEUE] attribute.

- int coda_raw_data

  The coda_raw_data variable is the pointer to the tail of the circular_buffer_raw_data[RAWDATAQUEUE] attribute.

- int coda_preprocessing

  The coda_preprocessing variable is the pointer to the tail of the circular_buffer_preprocessing[PREPROCESSINGQUEUE] attribute.

- int coda_visualization

  The coda_visualization variable is the pointer to the tail of the circular_buffer_visualization[VISUALIZATIONQUEUE] attribute.

- int go_general

  The go_general variable controls the main cycle of the threads.

- int go_preprocessing

  The go_preprocessing variable controls the main cycle of the threads.

- int go_raw_data

  The go_raw_data variable controls the main cycle of the threads.

- int go_visualization

  The go_visualization variable controls the main cycle of the threads.

- DigitizerObjectGeneric digitizer

  The digitizer variable represents the digitizer.

- RawData circular_buffer_raw_data [10]

  The circular_buffer_raw_data[RAWDATAQUEUE] variable is used to temporary store the data read from the digitizer.

- RawData circular_buffer_preprocessing [10]

  The circular_buffer_preprocessing[PREPROCESSINGQUEUE] variable is used to temporary store the data read from the digitizer.

- RawData circular_buffer_visualization [10]

  The circular_buffer_visualization[VISUALIZATIONQUEUE] variable is used to temporary store the data read from the digitizer.
- thread ∗ produttore_thread

  The thread id of the Produttore thread.
- thread ∗ consumatore_thread

  The thread id of the Consumatore thread.
- thread ∗ preprocessing_thread

  The thread id of the Preprocessing thread.
- thread ∗ visualization_thread

  The thread id of the Visualization thread.
- mutex ReservedConsumerDispatcherInputArea

  A mutex used to implement producer-consumer model.
- condition_variable BlockedProducerInput

  A cond variable used to implement producer-consumer model.
- condition_variable BlockedConsumerInput

  A cond variable used to implement producer-consumer model.
- mutex ReservedPreprocessingInputArea

  A mutex used to implement producer-consumer model.
- mutex ReservedVisualizationInputArea

  A mutex used to implement producer-consumer model.
- condition_variable Acquisition_Cond1

  A cond variable used to implement producer-consumer model.
- condition_variable Acquisition_Cond2

  A cond variable used to implement producer-consumer model.
- condition_variable raw_cond

  When the rawdata thread is not active, it is suspended in the raw_cond condition variable.
- condition_variable pre_cond

  When the preprocessing thread is not active, it is suspended in the pre_cond condition variable.
- condition_variable vis_cond

  When the visualization thread is not active, it is suspended in the vis_cond condition variable.
- int imstarted

  The imstarted variable indicates if the user has already used "start" command.
- int imset

  The imset variable indicates if the user has already init the object whith DigitizerStateMachineInit(const char ∗conf_-file) method.

5.10.1 Detailed Description

Definition at line 22 of file DigitizerStateMachine.h.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 DigitizerStateMachine::DigitizerStateMachine ( )

The DigitizerStateMachine(const char ∗) constructor sets go_general variable to zero.

The constructor is private in order to implement singleton design pattern.

Definition at line 359 of file DigitizerStateMachine.cpp.

5.10.3 Member Function Documentation

5.10.3.1 void DigitizerStateMachine::Consumatore_Dispatcher ( )

The Consumatore thread picks data from the circular_buffer_raw_data[RAWDATAQUEUE] and writes them into the hard disk.

Returns

void

Definition at line 154 of file DigitizerStateMachine.cpp.

5.10.3.2 void DigitizerStateMachine::DigitizerStateMachineInit (const char ∗ *conf_file*)

The DigitizerStateMachineInit(const char ∗) configures the digitizer.

Returns

void.

Parameters

conf_file	is the path of the configuration file.

Definition at line 370 of file DigitizerStateMachine.cpp.

5.10.3.3 void DigitizerStateMachine::DigitizerStateMachinePrintStatus ( )

The DigitizerStateMachinePrintStatus method prints some informations about the threads of the object.

Returns

void

Definition at line 606 of file DigitizerStateMachine.cpp.

5.10.3.4 void DigitizerStateMachine::DigitizerStateMachineQuit ( )

The DigitizerStateMachineQuit method closes the digitizer.

Returns

void

Definition at line 482 of file DigitizerStateMachine.cpp.

5.10.3.5 void DigitizerStateMachine::DigitizerStateMachineRawDataInit ( )

The DigitizerStateMachineRawDataInit method creates the Produttore and the Consumatore threads.

Il metodo DigitizerStateMachineRawDataInit fa riferimento ai parametri RAWDATAQUEUE, PREPROCESSINGQ-UEUE e VISUALIZATIONQUEUE definiti in DefineGeneral.h.

It also: initializes the mutexes and the cond variables; sets to 0 num_mex, testa_raw_data, coda_raw_data and sets to 1 the go_general variable.

Returns

void

Si consiglia di non superare il valore di 100 per non intasare le risorse del calcolatore.

Definition at line 411 of file DigitizerStateMachine.cpp.

5.10.3.6 void DigitizerStateMachine::DigitizerStateMachineSendSWTrigger ( )

The DigitizerStateMachineSendSWTrigger method sends one software trigger to the digitizer.

Returns

void

Definition at line 526 of file DigitizerStateMachine.cpp.

5.10.3.7 void DigitizerStateMachine::DigitizerStateMachineSetup (const char ∗ *conf_file*)

The DigitizerStateMachineSetup method sets the digitizer with the settings specified in the configuration file.

Il metodo DigitizerStateMachineSetup e' fondamentale perche' esegue tutte le funzioni per impostare il digitizer.

Parameters

conf_file	is the path of the configuration file

Returns

void

Definition at line 385 of file DigitizerStateMachine.cpp.

5.10.3.8 void DigitizerStateMachine::DigitizerStateMachineStartReading ( )

The DigitizerStateMachineStartReading method begins the data acquisition from the digitizer.

Returns

void

Definition at line 455 of file DigitizerStateMachine.cpp.

5.10.3.9 void DigitizerStateMachine::DigitizerStateMachineStopReading ( )

The DigitizerStateMachineStopReading method stops the data acquisition from the digitizer.

Returns

void

Definition at line 494 of file DigitizerStateMachine.cpp.

5.10.3.10 void DigitizerStateMachine::DigitizerStateStartPreprocessing ( )

The DigitizerStartPreprocessing method starts the preprocessing actions.

Returns

void

Definition at line 537 of file DigitizerStateMachine.cpp.

5.10.3.11 void DigitizerStateMachine::DigitizerStateStartRawDataWriting ( )

The DigitizerStartRawDataWriting method starts the visualization actions.

Returns

void

Definition at line 572 of file DigitizerStateMachine.cpp.

5.10.3.12 void DigitizerStateMachine::DigitizerStateStartVisualization ( )

The DigitizerStartVisualization method starts the visualization actions.

Returns

void

Definition at line 549 of file DigitizerStateMachine.cpp.

5.10.3.13 void DigitizerStateMachine::DigitizerStateStopPreprocessing ( )

The DigitizerStopPreprocessing method starts the preprocessing actions.

Returns

void

Definition at line 561 of file DigitizerStateMachine.cpp.

5.10.3.14 void DigitizerStateMachine::DigitizerStateStopRawDataWriting ( )

The DigitizerStopRawDataWriting method starts the preprocessing actions.

Returns

void

Definition at line 584 of file DigitizerStateMachine.cpp.

5.10.3.15 void DigitizerStateMachine::DigitizerStateStopVisualization ( )

The DigitizerStopVisualization method starts the visualization actions.

Returns

void

Definition at line 595 of file DigitizerStateMachine.cpp.

5.10.3.16 void DigitizerStateMachine::Preprocessing ( )

The Preprocessing thread picks data from the circular_buffer_preprocessing[PREPROCESSINGQUEUE] and performs the preprocessing data actions.

Returns

void

Definition at line 219 of file DigitizerStateMachine.cpp.

5.10.3.17 void DigitizerStateMachine::Produttore ( )

The Produttore thread reads data from the digitizer and puts them in the buffers.

Returns

void

Definition at line 29 of file DigitizerStateMachine.cpp.

5.10.3.18 void DigitizerStateMachine::Visualization ( )

The Visualization thread picks data from the circular_buffer_visualization[VISUALIZATIONQUEUE] and visualizes them using the gnuplot program.

Returns

void

Definition at line 281 of file DigitizerStateMachine.cpp.

5.10.4 Member Data Documentation

5.10.4.1 condition_variable DigitizerStateMachine::Acquisition_Cond1

A cond variable used to implement producer-consumer model.

Definition at line 267 of file DigitizerStateMachine.h.

5.10.4.2 condition_variable DigitizerStateMachine::Acquisition_Cond2

A cond variable used to implement producer-consumer model.

Definition at line 272 of file DigitizerStateMachine.h.

5.10.4.3 condition_variable DigitizerStateMachine::BlockedConsumerInput

A cond variable used to implement producer-consumer model.

Definition at line 252 of file DigitizerStateMachine.h.

5.10.4.4 condition_variable DigitizerStateMachine::BlockedProducerInput

A cond variable used to implement producer-consumer model.

Definition at line 247 of file DigitizerStateMachine.h.

5.10.4.5 RawData DigitizerStateMachine::circular_buffer_preprocessing[10]

The circular_buffer_preprocessing[PREPROCESSINGQUEUE] variable is used to temporary store the data read from the digitizer.

The data are put in the circular_buffer_preprocessing[PREPROCESSINGQUEUE] by Produttore thread and they are picked up by Preprocessing thread.

Definition at line 107 of file DigitizerStateMachine.h.

5.10.4.6 RawData DigitizerStateMachine::circular_buffer_raw_data[10]

The circular_buffer_raw_data[RAWDATAQUEUE] variable is used to temporary store the data read from the digitizer.

The data are put in the circular_buffer_raw_data[RAWDATAQUEUE] by Produttore thread and they are picked up by Consumatore thread.

Definition at line 101 of file DigitizerStateMachine.h.

5.10.4.7 RawData DigitizerStateMachine::circular_buffer_visualization[10]

The circular_buffer_visualization[VISUALIZATIONQUEUE] variable is used to temporary store the data read from the digitizer.

The data are put in the circular_buffer_visualization[VISUALIZATIONQUEUE] by Produttore thread and they are picked up by Visualization thread.

Definition at line 113 of file DigitizerStateMachine.h.

5.10.4.8 int DigitizerStateMachine::coda_preprocessing

The coda_preprocessing variable is the pointer to the tail of the circular_buffer_preprocessing[PREPROCESSING-QUEUE] attribute.

Definition at line 65 of file DigitizerStateMachine.h.

5.10.4.9 int DigitizerStateMachine::coda_raw_data

The coda_raw_data variable is the pointer to the tail of the circular_buffer_raw_data[RAWDATAQUEUE] attribute.

Definition at line 60 of file DigitizerStateMachine.h.

5.10.4.10 int DigitizerStateMachine::coda_visualization

The coda_visualization variable is the pointer to the tail of the circular_buffer_visualization[VISUALIZATIONQUEUE] attribute.

Definition at line 70 of file DigitizerStateMachine.h.

5.10.4.11 thread∗ DigitizerStateMachine::consumatore_thread

The thread id of the Consumatore thread.

Definition at line 227 of file DigitizerStateMachine.h.

5.10.4.12 DigitizerObjectGeneric DigitizerStateMachine::digitizer

The digitizer variable represents the digitizer.

Definition at line 95 of file DigitizerStateMachine.h.

5.10.4.13 int DigitizerStateMachine::go_general

The go_general variable controls the main cycle of the threads.

Definition at line 75 of file DigitizerStateMachine.h.

5.10.4.14 int DigitizerStateMachine::go_preprocessing

The go_preprocessing variable controls the main cycle of the threads.

Definition at line 80 of file DigitizerStateMachine.h.

5.10.4.15 int DigitizerStateMachine::go_raw_data

The go_raw_data variable controls the main cycle of the threads.

Definition at line 85 of file DigitizerStateMachine.h.

5.10.4.16 int DigitizerStateMachine::go_visualization

The go_visualization variable controls the main cycle of the threads.

Definition at line 90 of file DigitizerStateMachine.h.

5.10.4.17 int DigitizerStateMachine::imset

The imset variable indicates if the user has already init the object whith DigitizerStateMachineInit(const char ∗conf-_file) method.

Definition at line 304 of file DigitizerStateMachine.h.

5.10.4.18 int DigitizerStateMachine::imstarted

The imstarted variable indicates if the user has already used "start" command.

Definition at line 299 of file DigitizerStateMachine.h.

5.10.4.19 int DigitizerStateMachine::num_mex_preprocessing

The num_mex_preprocessing variable represents the number of messages stored in the circular_buffer_-preprocessing[PREPROCESSINGQUEUE] attribute.

Definition at line 35 of file DigitizerStateMachine.h.

5.10.4.20 int DigitizerStateMachine::num_mex_raw_data

The num_mex_raw_data variable represents the number of messages stored in the circular_buffer_raw_data[RA-WDATAQUEUE] attribute.

Definition at line 30 of file DigitizerStateMachine.h.

5.10.4.21 int DigitizerStateMachine::num_mex_visualization

The num_mex_visualization variable represents the number of messages stored in the circular_buffer_-visualization[VISUALIZATIONQUEUE] attribute.

Definition at line 40 of file DigitizerStateMachine.h.

5.10.4.22 condition_variable DigitizerStateMachine::pre_cond

When the preprocessing thread is not active, it is suspended in the pre_cond condition variable.

Definition at line 282 of file DigitizerStateMachine.h.

5.10.4.23 thread∗ DigitizerStateMachine::preprocessing_thread

The thread id of the Preprocessing thread.

Definition at line 232 of file DigitizerStateMachine.h.

5.10.4.24 thread∗ DigitizerStateMachine::produttore_thread

The thread id of the Produttore thread.

Definition at line 222 of file DigitizerStateMachine.h.

5.10.4.25 condition_variable DigitizerStateMachine::raw_cond

When the rawdata thread is not active, it is suspended in the raw_cond condition variable.

Definition at line 277 of file DigitizerStateMachine.h.

5.10.4.26 mutex DigitizerStateMachine::ReservedConsumerDispatcherInputArea

A mutex used to implement producer-consumer model.

Definition at line 242 of file DigitizerStateMachine.h.

5.10.4.27 mutex DigitizerStateMachine::ReservedPreprocessingInputArea

A mutex used to implement producer-consumer model.

Definition at line 257 of file DigitizerStateMachine.h.

5.10.4.28 mutex DigitizerStateMachine::ReservedVisualizationInputArea

A mutex used to implement producer-consumer model.

Definition at line 262 of file DigitizerStateMachine.h.

5.10.4.29 int DigitizerStateMachine::testa_preprocessing

The testa_preprocessing variable is the pointer to the head of the circular_buffer_preprocessing[PREPROCESSIN-GQUEUE] attribute.

Definition at line 50 of file DigitizerStateMachine.h.

5.10.4.30 int DigitizerStateMachine::testa_raw_data

The testa_raw_data variable is the pointer to the head of the circular_buffer_raw_data[RAWDATAQUEUE] attribute.

Definition at line 45 of file DigitizerStateMachine.h.

5.10.4.31 int DigitizerStateMachine::testa_visualization

The testa_visualization variable is the pointer to the head of the circular_buffer_visualization[VISUALIZATIONQU-EUE] attribute.

Definition at line 55 of file DigitizerStateMachine.h.

5.10.4.32 condition_variable DigitizerStateMachine::vis_cond

When the visualization thread is not active, it is suspended in the vis_cond condition variable.

Definition at line 287 of file DigitizerStateMachine.h.

5.10.4.33 thread∗ DigitizerStateMachine::visualization_thread

The thread id of the Visualization thread.

Definition at line 237 of file DigitizerStateMachine.h.

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

- DigitizerStateMachine.h
- DigitizerStateMachine.cpp

5.11 FastObject Class Reference

The FastObject class picks up all the settings taken from the configuration file regarding a TRn input channel of the digitizer (only for x742 series).

```
#include <ConfObject.h>
```

Public Member Functions

- FastObject ()

  The FastObject() constructor sets all the values of the fast_object attributes to -1.
- void PrintFast ()

  The PrintFast() method prints all the fast_object attributes.

Public Attributes

- int set

  The set variable indicates if a TRn has been set or not.
- int numFast

  The numFast variable indicates the number of the TRn.
- int dc_offset

  The variable dc_offset indicates the dc_offset being applicated to the TRn.
- int trigger_threshold

  The variable trigger_threshold contains the threshold for the TRn auto trigger.

5.11.1 Detailed Description

The FastObject class picks up all the settings taken from the configuration file regarding a TRn input channel of the digitizer (only for x742 series).

The number of the channel is indicated by numFast variable. If the channel has not been set, the set variable is set to -1 like the other variables.

Author

Daniele Berto

Definition at line 130 of file ConfObject.h.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 FastObject::FastObject ( )

The FastObject() constructor sets all the values of the fast_object attributes to -1.

Definition at line 78 of file ConfObject.cpp.

5.11.3 Member Function Documentation

5.11.3.1 void FastObject::PrintFast ( )

The PrintFast() method prints all the fast_object attributes.

Returns

void

Definition at line 124 of file ConfObject.cpp.

5.11.4 Member Data Documentation

5.11.4.1 int FastObject::dc_offset

The variable dc_offset indicates the dc_offset being applicated to the TRn.

Definition at line 147 of file ConfObject.h.

5.11.4.2 int FastObject::numFast

The numFast variable indicates the number of the TRn.

Definition at line 142 of file ConfObject.h.

5.11.4.3 int FastObject::set

The set variable indicates if a TRn has been set or not.

Definition at line 137 of file ConfObject.h.

5.11.4.4 int FastObject::trigger_threshold

The variable trigger_threshold contains the threshold for the TRn auto trigger.

Definition at line 152 of file ConfObject.h.

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

- ConfObject.h
- ConfObject.cpp

5.12 GroupObject Class Reference

The GroupObject class picks up all the settings taken from the configuration file regarding one group of the digitizer.

```
#include <ConfObject.h>
```

Public Member Functions

- GroupObject ()

  The GroupObject() constructor sets all the values of the group_object attributes to -1.
- void PrintGroup ()

  The PrintGroup() method prints all the group_object attributes.

Public Attributes

- int set

  The set variable indicates if a group has been set or not.
- int numGroup

  The numGroup variable indicates the number of the group.
- int enable_input

  The enable_input variable indicates if a group can receive input or not.
- int dc_offset

  The variable dc_offset indicates the dc_offset being applicated to the group.
- int trigger_threshold

  The variable trigger_threshold contains the threshold for the group auto trigger.
- int group_trg_enable_mask

  this option is used only for the Models x740.

5.12.1 Detailed Description

The GroupObject class picks up all the settings taken from the configuration file regarding one group of the digitizer.

The number of the group is indicated by numGroup variable. If the group has not been set, the set variable is set to -1 like the other variables.

Author

Daniele Berto

Definition at line 73 of file ConfObject.h.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 GroupObject::GroupObject ( )

The GroupObject() constructor sets all the values of the group_object attributes to -1.

Definition at line 67 of file ConfObject.cpp.

5.12.3 Member Function Documentation

5.12.3.1 void GroupObject::PrintGroup ( )

The PrintGroup() method prints all the group_object attributes.

Returns

void

Definition at line 106 of file ConfObject.cpp.

5.12.4 Member Data Documentation

5.12.4.1 int GroupObject::dc_offset

The variable dc_offset indicates the dc_offset being applicated to the group.

Definition at line 95 of file ConfObject.h.

5.12.4.2 int GroupObject::enable_input

The enable_input variable indicates if a group can receive input or not.

Definition at line 90 of file ConfObject.h.

5.12.4.3 int GroupObject::group_trg_enable_mask

this option is used only for the Models x740.

These models have the channels grouped 8 by 8; one group of 8 channels has a common trigger that is generated as the OR of the self trigger of the channels in the group that are enabled by this mask.

Definition at line 108 of file ConfObject.h.

5.12.4.4 int GroupObject::numGroup

The numGroup variable indicates the number of the group.

Definition at line 85 of file ConfObject.h.

5.12.4.5 int GroupObject::set

The set variable indicates if a group has been set or not.

Definition at line 80 of file ConfObject.h.

5.12.4.6 int GroupObject::trigger_threshold

The variable trigger_threshold contains the threshold for the group auto trigger.

Definition at line 100 of file ConfObject.h.

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

- ConfObject.h
- ConfObject.cpp

5.13 Input Class Reference

```
#include <Input.h>
```

Public Member Functions

- void GetInput (char ∗input_buffer_sending)

  The GetInput method copies the input from the stdin to the parameter input_buffer_sending.
- void Finish ()

  The Finish method shuts down the object threads.
- void Producer ()

  The Producer method gets the input from the stdin and puts it in the input_buffer attribute.
- Input ()

  The Input constructor sets num_mex to 0 and go to 1.

Public Attributes

- char ∗ input_buffer

  The input_buffer variable contains the informations shared by the threads.
- thread ∗ producer_thread

  The producer_thread gets the input from the stdin and puts it in the input_buffer attribute.
- mutex mutex1

  The mutex1 mutex guarantees the consistency of the informations retrieved by the get_input method.
- int go

  The go variable controls the threads.
- int num_mex

  The num_mex variable indicates if a message is ready to be picked by the get_input method.

5.13.1 Detailed Description

Definition at line 17 of file Input.h.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 Input::Input ( )

The Input constructor sets num_mex to 0 and go to 1.

It also initializates input_buffer and creates the producer thread.

Definition at line 32 of file Input.cpp.

5.13.3 Member Function Documentation

5.13.3.1 void Input::Finish ( )

The Finish method shuts down the object threads.

Returns

void

Definition at line 109 of file Input.cpp.

5.13.3.2 void Input::GetInput (char ∗ *input_buffer_sending*)

The GetInput method copies the input from the stdin to the parameter input_buffer_sending.

The caller of this method gets the user input: the method copies it in the parameter.

Parameters

input_buffer_-sending	is where the informations are copied

Returns

void

Definition at line 86 of file Input.cpp.

5.13.3.3 void Input::Producer ( )

The Producer method gets the input from the stdin and puts it in the input_buffer attribute.

Returns

void

Definition at line 44 of file Input.cpp.

5.13.4 Member Data Documentation

5.13.4.1 int Input::go

The go variable controls the threads.

Definition at line 53 of file Input.h.

5.13.4.2 char∗ Input::input_buffer

The input_buffer variable contains the informations shared by the threads.

Definition at line 24 of file Input.h.

5.13.4.3 mutex Input::mutex1

The mutex1 mutex guarantees the consistency of the informations retrieved by the get_input method.

Definition at line 48 of file Input.h.

5.13.4.4 int Input::num_mex

The num_mex variable indicates if a message is ready to be picked by the get_input method.

Definition at line 58 of file Input.h.

5.13.4.5 thread∗ Input::producer_thread

The producer_thread gets the input from the stdin and puts it in the input_buffer attribute.

Definition at line 43 of file Input.h.

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

- Input.h
- Input.cpp

5.14 LogFile Class Reference

The LogFile singleton is used to write the error codes retrieved by the CAENDigitizer function to the logfile indicates by the log_file_path attribute.

```
#include <LogFile.h>
```

Public Member Functions

- void LogFileSet (const char ∗log_file_arg)

  The LogFileSet method copies the log_file_arg parameter to the log_file_path and opens it using the log_file_punt file pointer.
- void LogFileWrite (CAEN_DGTZ_ErrorCode ret_arg, const char ∗file, const char ∗func, int line)

  The LogFileWrite method prints the meaning of the CAEN_DGTZ_ErrorCode ret_arg to the log file.
- void LogFileWriteString (const char ∗string)

  The LogFileWriteString method prints the string parameter to the log file.
- void LogFileRead ()

  The LogFileRead method prints the content of the log file.
- void LogFileMessageOn ()

  The LogFileMessageOn method sets to 1 the flag attribute allowing the object to print the message "Some error occurred".
- void LogFileMessageOff ()

  The LogFileMessageOff method sets to 0 the flag attribute disallowing the object to print the message "Some error occurred".

Static Public Member Functions

- static LogFile ∗ Instance ()

  The "Instance()" method returns a pointer to the LogFile instance.

Public Attributes

- const char ∗ log_file_path

  The log_file_path stores the path of the log file.

5.14.1 Detailed Description

The LogFile singleton is used to write the error codes retrieved by the CAENDigitizer function to the logfile indicates by the log_file_path attribute.

Using the "more" command, the user prints the content of the log file.

Author

Daniele Berto

Definition at line 13 of file LogFile.h.

5.14.2 Member Function Documentation

5.14.2.1 LogFile ∗ LogFile::Instance ( ) [static]

The "Instance()" method returns a pointer to the LogFile instance.

Definition at line 24 of file LogFile.cpp.

5.14.2.2 void LogFile::LogFileMessageOff ( )

The LogFileMessageOff method sets to 0 the flag attribute disallowing the object to print the message "Some error occurred".

Definition at line 43 of file LogFile.cpp.

5.14.2.3 void LogFile::LogFileMessageOn ( )

The LogFileMessageOn method sets to 1 the flag attribute allowing the object to print the message "Some error occurred".

Definition at line 34 of file LogFile.cpp.

5.14.2.4 void LogFile::LogFileRead ( )

The LogFileRead method prints the content of the log file.

Definition at line 109 of file LogFile.cpp.

5.14.2.5 void LogFile::LogFileSet (const char ∗ *log_file_arg*)

The LogFileSet method copies the log_file_arg parameter to the log_file_path and opens it using the log_file_punt file pointer.

Definition at line 51 of file LogFile.cpp.

5.14.2.6 void LogFile::LogFileWrite (CAEN_DGTZ_ErrorCode *ret_arg,* const char ∗ *file,* const char ∗ *func,* int *line*)

The LogFileWrite method prints the meaning of the CAEN_DGTZ_ErrorCode ret_arg to the log file.

Parameters

ret_arg	is the CAEN_DGTZ_ErrorCode being written on the log file.
file	is the path of the file in which the functions that generated the error code where located.
func	is the name of the function in which the functions that generated the error code where located.
line	is the line number of the functions that generated the error code.

Returns

void.

Definition at line 88 of file LogFile.cpp.

5.14.2.7 void LogFile::LogFileWriteString (const char ∗ *string*)

The LogFileWriteString method prints the string parameter to the log file.

Definition at line 78 of file LogFile.cpp.

5.14.3 Member Data Documentation

5.14.3.1 const char∗ LogFile::log_file_path

The log_file_path stores the path of the log file.

Definition at line 50 of file LogFile.h.

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

- LogFile.h
- LogFile.cpp

5.15 OutputModule Class Reference

The OutputModule class provides an useful way to manage the output of the program.

```
#include <OutputModule.h>
```

Public Member Functions

- void StdOutInsert (const char ∗string)

  The "StdOutInsert" method copies the parameter "string" to the "buffer" attribute.
- void StdOutInsertLex (char ∗string, int length)

  The "StdOutInsertLex" method copies the parameter "string" to the "buffer" attribute.
- void StdOutPrint ()

  The "StdOutPrint" method prints buffer to the stdout.
- void OutputModuleStdoutOn ()

  The "OutputModuleStdoutOn" method enables the object to print "buffer" to the stdout.
- void OutputModuleStdoutOff ()

  The "OutputModuleStdoutOff" method disables the object to print "buffer" to the stdout.
- void OutputModuleSockidOn (int sockid)

  The "OutputModuleSockidOn" method enables the object to send "buffer" to the TCP/IP user identificated by the "sockid" parameter.
- void OutputModuleSockidOff ()

  The "OutputModuleSockidOn" method disables the object to send "buffer" via TCP/IP.

- int TcpUserArrayInsert (int sockid)

  The "TcpUserArrayInsert" method inserts the parameter "sockid" in the "sockid_array".
- int TcpUserArrayDelete (int sockid)

  The "TcpUserArrayDelete" method deletes the parameter "sockid" from the "sockid_array".
- int TcpUserArraySendStdOut ()

  The "TcpUserArraySendStdOut" method sends the "buffer" attribute to the TCP/IP user identified by the output_-module_sockid attribute.
- void Output (const char ∗string)

  The "Output" method prints the "string" parameter to the stdout or send it via TCP/IP in accordance with the "output-_module_stdout" and the "output_module_sockid" (these variable could be modified by the "OutputModuleStdout-On"/"Off" and "OutputModuleSockidOn"/"Off" methods).
- void OutputFlex (const char ∗string, int length)

  The "OutputFlex" method prints the "string" parameter to the stdout or send it via TCP/IP in accordance with the "output_module_stdout" and the "output_module_sockid" (these variable could be modified by the "OutputModule-StdoutOn"/"Off" and "OutputModuleSockidOn"/"Off" methods).

Static Public Member Functions

- static OutputModule ∗ Instance ()

  The "Instance ()" method is used to implement the singleton design pattern.

Public Attributes

- ApplicationSetup ∗ output_module_application_setup

  The "output_module_application_setup" is a pointer to the ApplicationSetup singleton.
- char buffer [1000]

  The "buffer" array stores the output being printed.
- int sockid_array [100]

  The "sockid_array" array takes note of the users connected to the server via TCP/IP.

5.15.1 Detailed Description

The OutputModule class provides an useful way to manage the output of the program.

The program has to send output not only to stdout but also to the tcp users. The OutputModule class takes note of the output target with the "OutputModuleStdoutOn/Off" and the "OutputModuleSockidOn/Off" methods. It also takes note of the TCP users active on the server storing their sockid in sockid_array. It is possible to insert/delete a sockid with "TcpUserArrayInsert"/"Delete" methods.

Author

Daniele Berto

Definition at line 17 of file OutputModule.h.

5.15.2 Member Function Documentation

5.15.2.1 OutputModule ∗ OutputModule::Instance ( ) `[static]`

The "Instance ()" method is used to implement the singleton design pattern.

Definition at line 20 of file OutputModule.cpp.

5.15.2.2 void OutputModule::Output (const char ∗ *string*)

The "Output" method prints the "string" parameter to the stdout or send it via TCP/IP in accordance with the "output-_module_stdout" and the "output_module_sockid" (these variable could be modified by the "OutputModuleStdout-On"/"Off" and "OutputModuleSockidOn"/"Off" methods).

Definition at line 129 of file OutputModule.cpp.

5.15.2.3 void OutputModule::OutputFlex (const char ∗ *string*, int *length*)

The "OutputFlex" method prints the "string" parameter to the stdout or send it via TCP/IP in accordance with the "output_module_stdout" and the "output_module_sockid" (these variable could be modified by the "OutputModule-StdoutOn"/"Off" and "OutputModuleSockidOn"/"Off" methods).

The maximum number of characters being displayed is indicated by the "length" parameter. This method is used to send the information read from the configuration file by the lessical scanner.

Definition at line 138 of file OutputModule.cpp.

5.15.2.4 void OutputModule::OutputModuleSockidOff ( )

The "OutputModuleSockidOn" method disables the object to send "buffer" via TCP/IP.

Definition at line 164 of file OutputModule.cpp.

5.15.2.5 void OutputModule::OutputModuleSockidOn (int *sockid*)

The "OutputModuleSockidOn" method enables the object to send "buffer" to the TCP/IP user identificated by the "sockid" parameter.

The "sockid" parameter is copied to the "output_module_sockid" attribute.

Definition at line 159 of file OutputModule.cpp.

5.15.2.6 void OutputModule::OutputModuleStdoutOff ( )

The "OutputModuleStdoutOff" method disables the object to print "buffer" to the stdout.

Definition at line 155 of file OutputModule.cpp.

5.15.2.7 void OutputModule::OutputModuleStdoutOn ( )

The "OutputModuleStdoutOn" method enables the object to print "buffer" to the stdout.

Definition at line 151 of file OutputModule.cpp.

5.15.2.8 void OutputModule::StdOutInsert (const char ∗ *string*)

The "StdOutInsert" method copies the parameter "string" to the "buffer" attribute.

Definition at line 41 of file OutputModule.cpp.

5.15.2.9 void OutputModule::StdOutInsertLex (char ∗ *string*, int *length*)

The "StdOutInsertLex" method copies the parameter "string" to the "buffer" attribute.

Length indicates the dimension of "string".

Definition at line 48 of file OutputModule.cpp.

5.15.2.10 void OutputModule::StdOutPrint ( )

The "StdOutPrint" method prints buffer to the stdout.

Definition at line 55 of file OutputModule.cpp.

5.15.2.11 int OutputModule::TcpUserArrayDelete (int *sockid*)

The "TcpUserArrayDelete" method deletes the parameter "sockid" from the "sockid_array".

Definition at line 92 of file OutputModule.cpp.

5.15.2.12 int OutputModule::TcpUserArrayInsert (int *sockid*)

The "TcpUserArrayInsert" method inserts the parameter "sockid" in the "sockid_array".

Definition at line 75 of file OutputModule.cpp.

5.15.2.13 int OutputModule::TcpUserArraySendStdOut ( )

The "TcpUserArraySendStdOut" method sends the "buffer" attribute to the TCP/IP user identified by the output_-module_sockid attribute.

Definition at line 108 of file OutputModule.cpp.

5.15.3 Member Data Documentation

5.15.3.1 char OutputModule::buffer[1000]

The "buffer" array stores the output being printed.

It is filled by the "Output(const char ∗ string)" method.

Definition at line 59 of file OutputModule.h.

5.15.3.2 ApplicationSetup∗ OutputModule::output_module_application_setup

The "output_module_application_setup" is a pointer to the ApplicationSetup singleton.

Definition at line 49 of file OutputModule.h.

5.15.3.3 int OutputModule::sockid_array[100]

The "sockid_array" array takes note of the users connected to the server via TCP/IP.

Definition at line 100 of file OutputModule.h.

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

- OutputModule.h
- OutputModule.cpp

5.16 RawData Class Reference

The RawData class manages the readout from the digitizer.

```
#include <RawData.h>
```

Public Member Functions

- RawData ()

  The RawData constructor set "imset" to 0.
- void RawDataSet (DigitizerObjectGeneric digitizer_arg)

  The RawDataSet method copies the parameter handler_raw to the private attribute handler.
- void RawDataDel ()

  The RawDataDel method deallocates buffer attribute.
- void RawDataRead ()

  The RawDataRead method reads data from the digitizer and puts it in the buffer variable.
- void RawDataWriteOnFile (const char ∗file_arg)

  The RawWriteOnFile method writes the events read from the digitizer into the hard disk.
- void RawDataWriteOnFile (FILE ∗file, FILE ∗file_size)

  The RawWriteOnFile method writes the events read from the digitizer into the hard disk.
- void RawDataWriteDecodeEventOnPlotFile (const char ∗file_arg)

  The RawDataWriteDecodeEventOnPlotFile method decodes the events stored in the buffer attribute and writes the in the file specified by file_arg.
- void RawDataPlot (const char ∗file_arg, FILE ∗gnuplot)

  The RawDataPlot method plots the decodified events stored in the file specified by file_arg.
- void RawDataDecode ()

  The RawDataDecode method decodes the events stored in the buffer attributes.
- RawData & operator= (const RawData &p)

  This overloading is necessary in order to perform a deep copy of the object.

Public Attributes

- CAEN_DGTZ_ErrorCode ret

  The ret variable contains the error code returned by CAEN_DGTZ library functions.
- CAEN_DGTZ_EventInfo_t eventInfo

  The eventInfo variable contains the information about an event.
- DigitizerErrorObject ret_error

  The ret_error object is used to print the meaning of CAEN_DGTZ_ErrorCode.
- int handle

  The handle variable represents the device handler of the digitizer.
- int size

  The size variable represents the dimension of the readout buffer.
- CAEN_DGTZ_ReadMode_t Mode

  The Mode variable represents the mode of readout.
- int imset

  The imset variable indicates if the object has been set or not.
- FILE ∗ file

  The file pointer is used to write the raw data to the hard disk.
- int bsize

  The bsize variable represents the dimension of the data read from the digitizer.
- char ∗ buffer

  The buffer pointer contains the data read from the digitizer.
- DigitizerObjectGeneric digitizer

  The digitizer object is fundamental to get the informations to decode correctly the events read from the digitizer.

5.16.1 Detailed Description

The RawData class manages the readout from the digitizer.

Author

Daniele Berto

Definition at line 16 of file RawData.h.

5.16.2 Constructor & Destructor Documentation

5.16.2.1 RawData::RawData ( )

The RawData constructor set "imset" to 0.

Definition at line 48 of file RawData.cpp.

5.16.3 Member Function Documentation

5.16.3.1 RawData & RawData::operator= (const RawData & *p*)

This overloading is necessary in order to perform a deep copy of the object.

Now the assigment operator copies all the values of a RawData object, including the buffer attribute (not only the pointer of memory area).

Parameters

p	is the RawData being copied.

Returns

RawData &

Definition at line 23 of file RawData.cpp.

5.16.3.2 void RawData::RawDataDecode ( )

The RawDataDecode method decodes the events stored in the buffer attributes.

Returns

void

Definition at line 324 of file RawData.cpp.

5.16.3.3 void RawData::RawDataDel ( )

The RawDataDel method deallocates buffer attribute.

The method uses the free function from the stdlib, not the CAEN_DGTZ_FreeReadoutBuffer function.

Returns

void

Definition at line 67 of file RawData.cpp.

5.16.3.4 void RawData::RawDataPlot (const char ∗ *file_arg*, FILE ∗ *gnuplot*)

The RawDataPlot method plots the decodified events stored in the file specified by file_arg.

The method calls RawDataWriteDecodeEventOnPlotFile method. The code line "fflush(gnuplot)" is fundamental in order to send data to the gnuplot program in real-time.

Parameters

file_arg	is the path of the file being written with the raw data.
gnuplot	is the pipe communicating with the gnuplot program.

Returns

void

Definition at line 291 of file RawData.cpp.

5.16.3.5 void RawData::RawDataRead ( )

The RawDataRead method reads data from the digitizer and puts it in the buffer variable.

The method uses CAEN_DGTZ_ReadData function from CAENDigitizer library.

Returns

void

Definition at line 313 of file RawData.cpp.

5.16.3.6 void RawData::RawDataSet (DigitizerObjectGeneric *digitizer_arg*)

The RawDataSet method copies the parameter handler_raw to the private attribute handler.

It also allocates the buffer public attribute using MallocReadoutBuffer function.

Returns

void

Definition at line 56 of file RawData.cpp.

5.16.3.7 void RawData::RawDataWriteDecodeEventOnPlotFile (const char ∗ *file_arg*)

The RawDataWriteDecodeEventOnPlotFile method decodes the events stored in the buffer attribute and writes the in the file specified by file_arg.

The method is called by the RawDataPlot method.

Parameters

file_arg	is the path of file being written with the raw data.

Returns

void

Definition at line 111 of file RawData.cpp.

5.16.3.8 void RawData::RawDataWriteOnFile (const char * *file_arg*)

The RawWriteOnFile method writes the events read from the digitizer into the hard disk.

The method doesn't decode the events.

Parameters

file_arg	is the path of file being written with the raw data.

Returns

void

Definition at line 77 of file RawData.cpp.

5.16.3.9 void RawData::RawDataWriteOnFile (FILE ∗ *file*, FILE ∗ *file_size*)

The RawWriteOnFile method writes the events read from the digitizer into the hard disk.

The method doesn't decode the events.

Parameters

file	is the pointer to the file being written with the raw data.
file_size	is the pointer to the file being written with the raw data size.

Returns

void

Definition at line 100 of file RawData.cpp.

5.16.4 Member Data Documentation

5.16.4.1 int RawData::bsize

The bsize variable represents the dimension of the data read from the digitizer.

Definition at line 70 of file RawData.h.

5.16.4.2 char∗ RawData::buffer

The buffer pointer contains the data read from the digitizer.

Definition at line 75 of file RawData.h.

5.16.4.3 DigitizerObjectGeneric RawData::digitizer

The digitizer object is fundamental to get the informations to decode correctly the events read from the digitizer.

Definition at line 80 of file RawData.h.

5.16.4.4 CAEN_DGTZ_EventInfo_t RawData::eventInfo

The eventInfo variable contains the information about an event.

Definition at line 29 of file RawData.h.

5.16.4.5 FILE∗ RawData::file

The file pointer is used to write the raw data to the hard disk.

Definition at line 60 of file RawData.h.

5.16.4.6 int RawData::handle

The handle variable represents the device handler of the digitizer.

Definition at line 39 of file RawData.h.

5.16.4.7 int RawData::imset

The imset variable indicates if the object has been set or not.

Definition at line 55 of file RawData.h.

5.16.4.8 CAEN_DGTZ_ReadMode_t RawData::Mode

The Mode variable represents the mode of readout.

Definition at line 50 of file RawData.h.

5.16.4.9 CAEN_DGTZ_ErrorCode RawData::ret

The ret variable contains the error code returned by CAEN_DGTZ library functions.

Definition at line 24 of file RawData.h.

5.16.4.10 DigitizerErrorObject RawData::ret_error

The ret_error object is used to print the meaning of CAEN_DGTZ_ErrorCode.

Definition at line 34 of file RawData.h.

5.16.4.11 int RawData::size

The size variable represents the dimension of the readout buffer.

The dimension is determined by CAEN_DGTZ_MallocReadoutBuffer function.

Definition at line 45 of file RawData.h.

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

- RawData.h
- RawData.cpp

5.17 TcpUser Class Reference

The TcpUser class provides an useful way to store data about the users of the server.

```
#include <TcpUser.h>
```

Public Member Functions

- TcpUser ()

  The TcpUser constructor sets to 0 the attribute "command_sent_by_user", to -1 "register_address" and "register_-data" and to 0 first_parameter and second_parameter.

Public Attributes

- char command_sent_by_user

  The command_sent_by_user variable contains the informations about the command sent by the user.
- char first_parameter [1000]

  Some commands need more than one parameter to be executed (ex.
- char second_parameter [1000]

  The second_parameter array stores the second parameter of a composite command.
- int register_address

  The register_address variable stores the address of the register being written or being read.
- int register_data

  The register_data variable stores the data being written in the register indicated by register_address.
- int user_sockid

  The user_sockid variable contains the sockid of the user who sent the command.

5.17.1 Detailed Description

The TcpUser class provides an useful way to store data about the users of the server.

Author

Daniele Berto

Definition at line 13 of file TcpUser.h.

5.17.2 Constructor & Destructor Documentation

5.17.2.1 TcpUser::TcpUser ( )

The TcpUser constructor sets to 0 the attribute "command_sent_by_user", to -1 "register_address" and "register_-data" and to 0 first_parameter and second_parameter.

Definition at line 11 of file TcpUser.cpp.

5.17.3 Member Data Documentation

5.17.3.1 char TcpUser::command_sent_by_user

The command_sent_by_user variable contains the informations about the command sent by the user.

See "DefineGeneral.h" for other informations about the available commands.

Definition at line 21 of file TcpUser.h.

5.17.3.2 char TcpUser::first_parameter[1000]

Some commands need more than one parameter to be executed (ex.

write register address data). The first_parameter array stores the first parameter of a composite command.

Definition at line 27 of file TcpUser.h.

5.17.3.3 int TcpUser::register_address

The register_address variable stores the address of the register being written or being read.

Definition at line 37 of file TcpUser.h.

5.17.3.4 int TcpUser::register_data

The register_data variable stores the data being written in the register indicated by register_address.

Definition at line 42 of file TcpUser.h.

5.17.3.5 char TcpUser::second_parameter[1000]

The second_parameter array stores the second parameter of a composite command.

Definition at line 32 of file TcpUser.h.

5.17.3.6 int TcpUser::user_sockid

The user_sockid variable contains the sockid of the user who sent the command.

Definition at line 47 of file TcpUser.h.

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

- TcpUser.h
- TcpUser.cpp

Chapter 6

File Documentation

6.1 Analizzatore.h File Reference

This file contains the declarations of the functions used to scan the config file.

Functions

- int AnalizzaInit (ConfObject ∗mioconfig, const char ∗file)

  AnalizzaInit function scans the file indicated in the second parameter searching the informations to open the digitizer and puts them in the conf_object ∗ mioconfig.
- int AnalizzaSetup (ConfObject ∗mioconfig, const char ∗file)

  AnalizzaSetup function scans the file indicated in the second parameter searching the information to setup the digitizer and puts the informations in the conf_object ∗ mioconfig.
- int AnalizzaInitPrint (ConfObject ∗mioconfig, const char ∗file)

  AnalizzaInitPrint function scans the file indicated in the second parameter searching the information to open the digitizer and puts the informations in the conf_object ∗ mioconfig.
- int AnalizzaSetupPrint (ConfObject ∗mioconfig, const char ∗file)

  AnalizzaSetupPrint function scans the file indicated in the second parameter searching the information to setup the digitizer and puts the informations in the conf_object ∗ mioconfig.
- void AnalizzaPrint (const char ∗file)

  AnalizzaPrint function scans the file indicated in the second parameter searching the information to setup the digitizer and prints them to the stdout.

6.1.1 Detailed Description

This file contains the declarations of the functions used to scan the config file. These functions are implemented in analizzatore.c file (generated by flex).

Author

Daniele Berto

Definition in file Analizzatore.h.

6.1.2 Function Documentation

6.1.2.1 int AnalizzaInit (**ConfObject** ∗ *mioconfig,* const char ∗ *file*)

AnalizzaInit function scans the file indicated in the second parameter searching the informations to open the digitizer and puts them in the conf_object ∗ mioconfig.

It does not print recognized lines.

Parameters

mioconfig	is where the informations are stored.
file	is the path of the configuration file.

Returns

int

6.1.2.2 int AnalizzaInitPrint (ConfObject ∗ *mioconfig*, const char ∗ *file*)

AnalizzaInitPrint function scans the file indicated in the second parameter searching the information to open the digitizer and puts the informations in the conf_object ∗ mioconfig.

It prints recognized lines.

Parameters

mioconfig	is where the informations are stored.
file	is the path of the configuration file.

Returns

int

6.1.2.3 void AnalizzaPrint (const char ∗ *file*)

AnalizzaPrint function scans the file indicated in the second parameter searching the information to setup the digitizer and prints them to the stdout.

Parameters

file	is the path of the configuration file.

Returns

void

6.1.2.4 int AnalizzaSetup (ConfObject ∗ *mioconfig*, const char ∗ *file*)

AnalizzaSetup function scans the file indicated in the second parameter searching the information to setup the digitizer and puts the informations in the conf_object ∗ mioconfig.

It does not print recognized lines.

Parameters

mioconfig	is where the informations are stored.
file	is the path of the configuration file.

Returns

int

6.1.2.5 int AnalizzaSetupPrint (ConfObject ∗ *mioconfig*, const char ∗ *file*)

AnalizzaSetupPrint function scans the file indicated in the second parameter searching the information to setup the digitizer and puts the informations in the conf_object ∗ mioconfig.

It prints recognized lines.

Parameters

mioconfig	is where the informations are stored.
file	is the path of the configuration file.

Returns

int

6.2 Analizzatore.h

```

int AnalizzaInit (ConfObject * mioconfig, const char *file);

int AnalizzaSetup (ConfObject * mioconfig, const char *file);

int AnalizzaInitPrint (ConfObject * mioconfig, const char *file);

int AnalizzaSetupPrint (ConfObject * mioconfig, const char *file);

void AnalizzaPrint (const char *file);
```

6.3 AnalizzatoreUtils.c File Reference

This file contains the implementation of the utility functions used by the scanner.

```
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <regex.h>
#include <stdbool.h>
#include "DefineGeneral.h"
#include "ConfObject.h"
```

Functions

- bool reg_matches (const char ∗str, const char ∗pattern)

  La funzione reg_matches serve per comparare una stringa con una espressione regolare. Restituisce true se e' stata trovata l'espressione regolare pattern nella stringa str.
- char ∗ FindPointer (char ∗yytext)

  The FindPointer function returns a pointer to the first useful character after the command.
- int FindIntegerValue (char ∗yytext)

  The FindIntegerValue function returns the value of the first integer after the command.
- int OutputFileFormat (char ∗yytext)

  The OutputFileFormat function returns an integer that represents the format of the output file of the application.
- int OutputRisingFalling (char ∗yytext)

  The OutputRisingFalling function returns an integer that decides whether the trigger occurs on the rising or falling edge of the signal.
- int OutputNIMTTL (char ∗yytext)

  The OutputNIMTTL function returns an integer that represents the type of the front panel I/O LEMO connectors.
- int YesNoAnswer (char ∗yytext)

  The YesNoAnswer function returns an integer that represents yes or no.
- int GetAcquisitionMode (char ∗yytext)

  The GetAcquisitionMode function returns an integer that represents the acquisition mode.
- void GetOpenInformation (char ∗yytext, ConfObject ∗mioconfig)

The GetOpenInformation function gets the information for opening the digitizer from the yytext string and puts it in a ConfObject.

- void GetWriteRegisterInformation (char ∗yytext, ConfObject ∗mioconfig)

  The GetWriteRegisterInformation function gets the information for writing a specific register of the digitizer from the yytext string and puts it in a ConfObject.
- void ChInformation (char ∗yytext, ConfObject ∗mioconfig)

  The ChInformation function gets the information for setting a specific channel of the digitizer from the yytext string and puts it in a ConfObject.
- void GroupInformation (char ∗yytext, ConfObject ∗mioconfig)

  The GroupInformation function gets the information for setting a specific group of the digitizer from the yytext string and puts it in a ConfObject.
- void AllInformation (char ∗yytext, ConfObject ∗mioconfig)

  The AllInformation function gets the information for setting all the channels of the digitizer from the yytext string and puts it in a ConfObject.
- void FastInformation (char ∗yytext, ConfObject ∗mioconfig)

  The FastInformation function gets the information for setting TRn channels of the digitizer from the yytext string and puts it in a ConfObject.
- void ChannelEnableMask (char ∗yytext, ConfObject ∗mioconfig)

  The ChannelEnableMask function gets the information for enabling or not the input from the channels.
- void GroupEnableMask (char ∗yytext, ConfObject ∗mioconfig)

  The GroupEnableMask function gets the information for enabling or not the input from the groups.
- void ChannelTriggerEnableMask (char ∗yytext, ConfObject ∗mioconfig)

  The ChannelTriggerEnableMask function gets the information for setting TRn channels of the digitizer from the yytext string and puts it in a ConfObject.

6.3.1 Detailed Description

This file contains the implementation of the utility functions used by the scanner. Il file AnalizzatoreLessicale.-flex puo' generare, usando flex, uno scanner per il file di configurazione. Il contenuto delle stringhe riconosciute deve essere poi estrapolato e messo nel ConfObject: le funzioni seguenti eseguono questo compito. Ogni funzione esegue un compito molto specifico: sono progettate guardando alla singola stringa di configurazione da cui ricavare il contenuto.

Author

Daniele Berto

Definition in file AnalizzatoreUtils.c.

6.3.2 Function Documentation

6.3.2.1 void AllInformation (char ∗ *yytext,* ConfObject ∗ *mioconfig*)

The AllInformation function gets the information for setting all the channels of the digitizer from the yytext string and puts it in a ConfObject.

Ex. yytext contains the string "ALL TRIGGER_THRESHOLD 0x10000000", so we can call AllInformation (yytext, mioconfig).

Parameters

yytext	contains a matched string.

mioconfig	is where the retrieved informations are stored.

Returns

void

Definition at line 347 of file AnalizzatoreUtils.c.

6.3.2.2 void ChannelEnableMask (char ∗ *yytext,* ConfObject ∗ *mioconfig*)

The ChannelEnableMask function gets the information for enabling or not the input from the channels.

It gets them from the yytext string and puts it in a ConfObject.

Ex. yytext contains the string "CHANNEL_ENABLE_MASK 0x00000111", so we can call ChannelEnableMask (yytext, mioconfig).

Parameters

yytext	contains a matched string.
mioconfig	is where the retrieved informations are stored.

Returns

void

Definition at line 447 of file AnalizzatoreUtils.c.

6.3.2.3 void ChannelTriggerEnableMask (char ∗ *yytext,* ConfObject ∗ *mioconfig*)

The ChannelTriggerEnableMask function gets the information for setting TRn channels of the digitizer from the yytext string and puts it in a ConfObject.

Ex. yytext contains the string "FAST TRIGGER_THRESHOLD 0x10000000", so we can call ChannelTriggerEnable-Mask (yytext, mioconfig).

Parameters

yytext	contains a matched string.
mioconfig	is where the retrieved informations are stored.

Returns

void

Definition at line 499 of file AnalizzatoreUtils.c.

6.3.2.4 void ChInformation (char ∗ *yytext,* ConfObject ∗ *mioconfig*)

The ChInformation function gets the information for setting a specific channel of the digitizer from the yytext string and puts it in a ConfObject.

Ex. yytext contains the string "CH 1 CHANNEL_TRIGGER DISABLED", so we can call ChInformation (yytext, mioconfig).

Parameters

yytext	contains a matched string.
mioconfig	is where the retrieved informations are stored.

Returns

void

Definition at line 183 of file AnalizzatoreUtils.c.

6.3.2.5 void FastInformation (char ∗ *yytext,* ConfObject ∗ *mioconfig*)

The FastInformation function gets the information for setting TRn channels of the digitizer from the yytext string and puts it in a ConfObject.

Ex. yytext contains the string "FAST TRIGGER_THRESHOLD 0x10000000", so we can call FastInformation (yytext, mioconfig).

Parameters

yytext	contains a matched string.
mioconfig	is where the retrieved informations are stored.

Returns

void

Definition at line 401 of file AnalizzatoreUtils.c.

6.3.2.6 int FindIntegerValue (char ∗ *yytext*)

The FindIntegerValue function returns the value of the first integer after the command.

Ex: the string "POST_TRIGGER 20" is in yytext and we call value = find_integer_value(yytext). After that value contains contains the integer 20.

Parameters

yytext	contains a matched string.

Returns

int

Definition at line 51 of file AnalizzatoreUtils.c.

6.3.2.7 char∗ FindPointer (char ∗ *yytext*)

The FindPointer function returns a pointer to the first useful character after the command.

Ex: the string "POST_TRIGGER 20" is in yytext and we call punt = find_pointer(yytext). After that punt contains a pointer to the character ’2’. This function is used by many other functions of this file.

Parameters

yytext	contains a matched string.

Returns

char

Definition at line 38 of file AnalizzatoreUtils.c.

6.3.2.8 int GetAcquisitionMode (char ∗ *yytext*)

The GetAcquisitionMode function returns an integer that represents the acquisition mode.

Values: 0 = ACQUISITION_ONLY, 1 = ACQUISITION_AND_TRGOUT, 2 = DISABLED, other=UNSET.

Parameters

yytext	contains a matched string.

Returns

int

Definition at line 113 of file AnalizzatoreUtils.c.

6.3.2.9 void GetOpenInformation (char ∗ *yytext,* ConfObject ∗ *mioconfig*)

The GetOpenInformation function gets the information for opening the digitizer from the yytext string and puts it in a ConfObject.

Ex. yytext contains the string "OPEN PCI 0 0 0x11110000", so we can call GetOpenInformation (yytext, mioconfig).

Parameters

yytext	contains a matched string.
mioconfig	is where the retrieved informations are stored.

Returns

void

Definition at line 132 of file AnalizzatoreUtils.c.

6.3.2.10 void GetWriteRegisterInformation (char ∗ *yytext,* ConfObject ∗ *mioconfig*)

The GetWriteRegisterInformation function gets the information for writing a specific register of the digitizer from the yytext string and puts it in a ConfObject.

Ex. yytext contains the string "WRITE_REGISTER 1080 0000 0100", so we can call GetWriteRegisterInformation (yytext, mioconfig).

Parameters

yytext	contains a matched string.
mioconfig	is where the retrieved informations are stored.

Returns

void

Definition at line 162 of file AnalizzatoreUtils.c.

6.3.2.11 void GroupEnableMask (char ∗ *yytext,* ConfObject ∗ *mioconfig*)

The GroupEnableMask function gets the information for enabling or not the input from the groups.

It gets them from the yytext string and puts it in a ConfObject.

Ex. yytext contains the string "GROUP_ENABLE_MASK 0x10000000", so we can call GroupEnableMask (yytext, mioconfig).

Parameters

yytext	contains a matched string.
mioconfig	is where the retrieved informations are stored.

Returns

void

Definition at line 473 of file AnalizzatoreUtils.c.

6.3.2.12 void GroupInformation (char ∗ *yytext,* ConfObject ∗ *mioconfig*)

The GroupInformation function gets the information for setting a specific group of the digitizer from the yytext string and puts it in a ConfObject.

Ex. yytext contains the string "GR 1 CHANNEL_TRIGGER DISABLED", so we can call GroupInformation (yytext, mioconfig).

Parameters

yytext	contains a matched string.
mioconfig	is where the retrieved informations are stored.

Returns

void

Definition at line 277 of file AnalizzatoreUtils.c.

6.3.2.13 int OutputFileFormat (char ∗ *yytext*)

The OutputFileFormat function returns an integer that represents the format of the output file of the application.

See ConfObject documentation for other informations.

Parameters

yytext	contains a matched string.

Returns

int

Definition at line 61 of file AnalizzatoreUtils.c.

6.3.2.14 int OutputNIMTTL (char ∗ *yytext*)

The OutputNIMTTL function returns an integer that represents the type of the front panel I/O LEMO connectors.

See ConfObject documentation for other informations.

Parameters

yytext	contains a matched string.

Returns

int

Definition at line 87 of file AnalizzatoreUtils.c.

6.3.2.15 int OutputRisingFalling (char * *yytext*)

The OutputRisingFalling function returns an integer that decides whether the trigger occurs on the rising or falling edge of the signal.

See ConfObject documentation for other informations.

Parameters

yytext	contains a matched string.

Returns

int

Definition at line 74 of file AnalizzatoreUtils.c.

6.3.2.16 bool reg_matches (const char * *str,* const char * *pattern*)

La funzione reg_matches serve per comparare una stringa con una espressione regolare. Restituisce true se e' stata trovata l'espressione regolare pattern nella stringa str.

Definition at line 20 of file AnalizzatoreUtils.c.

6.3.2.17 int YesNoAnswer (char * *yytext*)

The YesNoAnswer function returns an integer that represents yes or no.

Values: 0 = NO, 1 = YES, other=UNSET.

Parameters

yytext	contains a matched string.

Returns

int

Definition at line 100 of file AnalizzatoreUtils.c.

6.4 AnalizzatoreUtils.c

```
00001
00010 #include <stdlib.h>
00011 #include <string.h>
00012 #include <assert.h>
00013 #include <regex.h>
00014 #include <stdbool.h>
00015 #include "DefineGeneral.h"
00016 #include "ConfObject.h"
```

```c

bool
reg_matches (const char *str, const char *pattern)
{
  regex_t re;
  int ret;

  if (regcomp (&re, pattern, REG_EXTENDED) != 0)
    return false;

  ret = regexec (&re, str, (size_t) 0, NULL, 0);
  regfree (&re);

  if (ret == 0)
    return true;

  return false;
}

char *
FindPointer (char *yytext)
{
  int i = 0;
  char *punt;
  while (yytext[i] != ' ' && yytext[i] != '\t')
    i++;
  while (yytext[i] == ' ' || yytext[i] == '\t')
    i++;
  punt = yytext + i;
  return punt;
}

int
FindIntegerValue (char *yytext)
{
  char *punt;
  int integer_value;
  punt = FindPointer (yytext);
  integer_value = atoi (punt);
  return integer_value;
}

int
OutputFileFormat (char *yytext)
{
  char *punt;
  punt = FindPointer (yytext);
  if (reg_matches (punt, "[Bb][Ii][Nn][Aa][Rr][Yy]"))
    return 0;
  else if (reg_matches (punt, "[Aa][Ss][Cc][Ii][Ii]"))
    return 1;
  else
    return -1;
}

int
OutputRisingFalling (char *yytext)
{
  char *punt;
  punt = FindPointer (yytext);
  if (reg_matches (punt, "[rR][iI][sS][iI][nN][gG]"))
    return 0;
  else if (reg_matches (punt, "[fF][aA][lL][lL][iI][nN][gG]"))
    return 1;
  else
    return -1;
}

int
OutputNIMTTL (char *yytext)
{
  char *punt;
  punt = FindPointer (yytext);
  if (reg_matches (punt, "[Nn][Ii][Mm]"))
    return 0;
  else if (reg_matches (punt, "[Tt][Tt][Ll]"))
    return 1;
  else
    return -1;
}

int
YesNoAnswer (char *yytext)
{
  char *punt;
  punt = FindPointer (yytext);
  if (reg_matches (punt, "[Yy][Ee][Ss]"))
```

```
    return 1;
  else if (reg_matches (punt, "[Nn][Oo]"))
    return 0;
  else
    return -1;
}

int
GetAcquisitionMode (char *yytext)
{
  char *punt;
  punt = FindPointer (yytext);
  if (reg_matches
      (punt, "[aA][cC][qQ][uU][iI][sS][iI][tT][iI][oO][nN]_[oO][nN][lL][yY]"))
    return 0;
  else
    if (reg_matches
    (punt,
     "[aA][cC][qQ][uU][iI][sS][iI][tT][iI][oO][nN]_[aA][nN][dD]_[tT][rR][gG][oO][uU][tT]"))
    return 1;
  else if (reg_matches (punt, "[dD][iI][sS][aA][bB][lL][eE][dD]"))
    return 2;
  else
    return -1;
}

void
GetOpenInformation (char *yytext, ConfObject * mioconfig)
{
  char *punt;
  int Type;
  int LinkNumber;
  int NodeNumber;
  int BaseAddress;
  punt = FindPointer (yytext);
  if (reg_matches (punt, "[Uu][Ss][Bb]"))
    Type = 0;
  else if (reg_matches (punt, "[Pp][Cc][Ii]"))
    Type = 1;
  else
    Type = -1;
  LinkNumber = FindIntegerValue (punt);
  punt = FindPointer (punt);
  NodeNumber = FindIntegerValue (punt);
  punt = FindPointer (punt);
  punt = FindPointer (punt);
  BaseAddress = strtoul (punt, NULL, 16);

  mioconfig->LinkType = Type;
  mioconfig->LinkNumber = LinkNumber;
  mioconfig->ConetNode = NodeNumber;
  mioconfig->VMEBaseAddress = BaseAddress;

  //printf ("%d %d %d %d\n", Type, LinkNumber, NodeNumber, BaseAddress);
}

void
GetWriteRegisterInformation (char *yytext, ConfObject * mioconfig)
{
  char *punt;
  int Address;
  int Data;
  int Mask;
  punt = FindPointer (yytext);
  Address = strtoul (punt, NULL, 16);
  punt = FindPointer (punt);
  Data = strtoul (punt, NULL, 16);
  punt = FindPointer (punt);
  Mask = strtoul (punt, NULL, 16);
  mioconfig->Address_register = Address;
  mioconfig->Data_register = Data;
  mioconfig->Mask_register = Mask;
}


//**********************************************

void
ChInformation (char *yytext, ConfObject * mioconfig)
{
  char *punt;
  int mod;
  int yes_no;
  int channel;
  int channel_trigger;


  channel = FindIntegerValue (yytext);


  if (channel < MAXCHANNELOBJECT)
    {
      mioconfig->channels[channel].set = 1;
      mioconfig->channels[channel].numChannel = channel;

      punt = FindPointer (yytext);
      punt = FindPointer (punt);


      if (reg_matches (punt, "[Ee][Nn][Aa][Bb][Ll][Ee]_[iI][nN][pP][uU][tT]"))
        mod = 0;
      else if (reg_matches (punt, "[dD][cC]_[oO][fF][fF][sS][eE][tT]"))
        mod = 1;
      else if (reg_matches (punt, "
[tT][rR][iI][gG][gG][eE][rR]_[tT][hH][rR][eE][sS][hH][oO][lL][dD]"))
        mod = 2;
      else if (reg_matches
         (punt, "[cC][hH][aA][nN][nN][eE][lL]_[tT][rR][iI][gG][gG][eE][rR]"))
        mod = 3;
      else
        mod = -1;


      if (mod == 0)
        {                 //ENABLE_INPUT
          punt = FindPointer (punt);
          if (reg_matches (punt, "[Yy][Ee][Ss]"))
        {
          yes_no = 1;
          mioconfig->channels[channel].enable_input = yes_no;
        }
          else if (reg_matches (punt, "[Nn][Oo]"))
        {
          yes_no = 0;
          mioconfig->channels[channel].enable_input = yes_no;
        }
        }

      else if (mod == 1)
        {                 //DC_OFFSET
          punt = FindPointer (punt);
          int dc_offset = strtoul (punt, NULL, 16);
          mioconfig->channels[channel].dc_offset = dc_offset;
        }

      else if (mod == 2)
        {                 //TRIGGER_THRESHOLD
          punt = FindPointer (punt);
          int trigger_threshold = strtoul (punt, NULL, 16);
          mioconfig->channels[channel].trigger_threshold = trigger_threshold;
          //mioconfig->channels[channel].trigger_threshold;
        }

      else if (mod == 3)
        {                 //CHANNEL_TRIGGER
          punt = FindPointer (punt);
          if (reg_matches
           (punt,
            "[aA][cC][qQ][uU][iI][sS][iI][tT][iI][oO][nN]_[oO][nN][lL][yY]"))
        {
          channel_trigger = 0;
          mioconfig->channels[channel].channel_trigger = channel_trigger;
        }
          else
        if (reg_matches
            (punt,
             "[aA][cC][qQ][uU][iI][sS][iI][tT][iI][oO][nN]_[aA][nN][dD]_[tT][rR][gG][oO][uU][tT]"))
        {
          channel_trigger = 1;
          mioconfig->channels[channel].channel_trigger = channel_trigger;
        }
          else if (reg_matches (punt, "[dD][iI][sS][aA][bB][lL][eE][dD]"))
        {
          channel_trigger = 2;
          mioconfig->channels[channel].channel_trigger = channel_trigger;
        }
        } // else if (mod == 3)
      } //if (channel < MAXCHANNELOBJECT)
}  //END_FUNCTION

//**************************************************

void
GroupInformation (char *yytext, ConfObject * mioconfig)
```

```
{
  char *punt;
  int mod;
  int yes_no;
  int group;
  //int group_mask;

  group = FindIntegerValue (yytext);

  if (group < MAXGROUPOBJECT)
    {

      mioconfig->groups[group].set = 1;
      mioconfig->groups[group].numGroup = group;

      punt = FindPointer (yytext);
      punt = FindPointer (punt);
      if (reg_matches (punt, "[Ee][Nn][Aa][Bb][Ll][Ee]_[iI][nN][pP][uU][tT]"))
        mod = 0;
      else if (reg_matches (punt, "[dD][cC]_[oO][fF][fF][sS][eE][tT]"))
        mod = 1;
      else if (reg_matches (punt, "
[tT][rR][iI][gG][gG][eE][rR]_[tT][hH][rR][eE][sS][hH][oO][lL][dD]"))
        mod = 2;
      else if (reg_matches (punt, "
[gG][rR][oO][uU][pP]_[tT][rR][gG]_[eE][nN][aA][bB][lL][eE]_[mM][aA][sS][kK]"))
        mod = 3;
      else
        mod = -1;

      if (mod == 0)
        {                 //ENABLE_INPUT
          punt = FindPointer (punt);
          if (reg_matches (punt, "[Yy][Ee][Ss]"))
        {
          yes_no = 1;
          mioconfig->groups[group].enable_input = yes_no;
        }
          else if (reg_matches (punt, "[Nn][Oo]"))
        {
          yes_no = 0;
          mioconfig->groups[group].enable_input = yes_no;
        }
        }

      else if (mod == 1)
        {                 //DC_OFFSET
          punt = FindPointer (punt);
          int dc_offset = strtoul (punt, NULL, 16);
          mioconfig->groups[group].dc_offset = dc_offset;
        }

      else if (mod == 2)
        {                 //TRIGGER_THRESHOLD
          punt = FindPointer (punt);
          int trigger_threshold = strtoul (punt, NULL, 16);
          mioconfig->groups[group].trigger_threshold = trigger_threshold;
        }

      else if (mod == 3)
        {                 //GROUP_TRG_ENABLE_MASK
          punt = FindPointer (punt);
          int group_mask = strtoul (punt, NULL, 16);
          mioconfig->groups[group].group_trg_enable_mask = group_mask;
        } // else if (mod == 3)
    } // if (group < MAXGROUPOBJECT)
}  //END_FUNCTION

//************************************************

void
AllInformation (char *yytext, ConfObject * mioconfig)
{
  char *punt;
  int mod;
  int yes_no;
  punt = FindPointer (yytext);
  if (reg_matches (punt, "[Ee][Nn][Aa][Bb][Ll][Ee]_[iI][nN][pP][uU][tT]"))
    mod = 0;
  else if (reg_matches (punt, "[dD][cC]_[oO][fF][fF][sS][eE][tT]"))
    mod = 1;
  else
    if (reg_matches
    (punt,
     "[tT][rR][iI][gG][gG][eE][rR]_[tT][hH][rR][eE][sS][hH][oO][lL][dD]"))
    mod = 2;
  else
```

```c
    mod = -1;

  if (mod == 0)
    {               //ENABLE_INPUT

      punt = FindPointer (punt);
      if (reg_matches (punt, "[Yy][Ee][Ss]"))
    {
      yes_no = 1;
      mioconfig->enable_input = yes_no;
    }
      else if (reg_matches (punt, "[Nn][Oo]"))
    {
      yes_no = 0;
      mioconfig->enable_input = yes_no;
    }

    }

  else if (mod == 1)
    {               //DC_OFFSET
      punt = FindPointer (punt);
      int dc_offset = strtoul (punt, NULL, 16);
      mioconfig->dc_offset = dc_offset;
    }

  else if (mod == 2)
    {               //TRIGGER_THRESHOLD
      punt = FindPointer (punt);
      int trigger_threshold = strtoul (punt, NULL, 16);
      mioconfig->trigger_threshold = trigger_threshold;
    }

}               //END_FUNCTION

//************************************************

void
FastInformation (char *yytext, ConfObject * mioconfig)
{
  char *punt;
  int mod;
  //int yes_no;
  int fast;
  fast = FindIntegerValue (yytext);

    if (fast < MAXFASTOBJECT)
        {

  mioconfig->fasts[fast].set = 1;
  mioconfig->fasts[fast].numFast = fast;


  punt = FindPointer (yytext);
  punt = FindPointer (punt);

  if (reg_matches (punt, "[dD][cC]_[oO][fF][fF][sS][eE][tT]"))
    mod = 0;
  else
    if (reg_matches
    (punt,
     "[tT][rR][iI][gG][gG][eE][rR]_[tT][hH][rR][eE][sS][hH][oO][lL][dD]"))
    mod = 1;
  else
    mod = -1;

  if (mod == 0)
    {               //DC_OFFSET
      punt = FindPointer (punt);
      int dc_offset = strtoul (punt, NULL, 16);
      mioconfig->fasts[fast].dc_offset = dc_offset;
    }

  else if (mod == 1)
    {               //TRIGGER_THRESHOLD
      punt = FindPointer (punt);
      int trigger_threshold = strtoul (punt, NULL, 16);
      mioconfig->fasts[fast].trigger_threshold = trigger_threshold;
    }
        } // if (fast < MAXFASTOBJECT)

}               //END_FUNCTION

void
ChannelEnableMask (char *yytext, ConfObject * mioconfig)
{

  int i;
  unsigned int bit = 1;
  unsigned int maschera;

  char *punt;
  int channel_enable_mask;
  punt = FindPointer (yytext);
  mioconfig->channel_enable_mask = strtoul (punt, NULL, 16);

  maschera = (unsigned int) mioconfig->channel_enable_mask;
  for (i = 0; i < MAXCHANNELOBJECT; i++)
    {
      if (maschera & bit)
    {
      mioconfig->channels[i].enable_input = 1;
      mioconfig->channels[i].set = 1;
      mioconfig->channels[i].numChannel = i;
    }
      maschera >>= 1;
    }
}

void
GroupEnableMask (char *yytext, ConfObject * mioconfig)
{

  int i;
  unsigned int bit = 1;
  unsigned int maschera;

  char *punt;
  int channel_enable_mask;
  punt = FindPointer (yytext);
  mioconfig->group_enable_mask = strtoul (punt, NULL, 16);

  maschera = (unsigned int) mioconfig->group_enable_mask;
  for (i = 0; i < MAXGROUPOBJECT; i++)
    {
      if (maschera & bit)
    {
      mioconfig->groups[i].enable_input = 1;
      mioconfig->groups[i].set = 1;
      mioconfig->groups[i].numGroup = i;
    }
      maschera >>= 1;
    }
}

void
ChannelTriggerEnableMask (char *yytext, ConfObject * mioconfig)
{

  char *punt;
  int channel_trigger_enable_mask;
  punt = FindPointer (yytext);
  mioconfig->self_trigger_enable_mask = strtoul (punt, NULL, 16);
  punt = FindPointer (yytext);

  if (reg_matches
      (punt, "[aA][cC][qQ][uU][iI][sS][iI][tT][iI][oO][nN]_[oO][nN][lL][yY]"))
    {
      mioconfig->self_trigger_enable_mask_mode = 0;
    }
  else
    if (reg_matches
    (punt,
     "[aA][cC][qQ][uU][iI][sS][iI][tT][iI][oO][nN]_[aA][nN][dD]_[tT][rR][gG][oO][uU][tT]"))
    {
      mioconfig->self_trigger_enable_mask_mode = 1;
    }
  else if (reg_matches (punt, "[dD][iI][sS][aA][bB][lL][eE][dD]"))
    {
      mioconfig->self_trigger_enable_mask_mode = 2;
    }
}
```

6.5 AnalizzatoreUtils.h File Reference

This file contains the declaration of the utility functions used by the scanner.

```
#include <stdlib.h>
#include <string.h>
#include <assert.h>
```

Functions

- char ∗ FindPointer (char ∗yytext)

  The FindPointer function returns a pointer to the first useful character after the command.
- int FindIntegerValue (char ∗yytext)

  The FindIntegerValue function returns the value of the first integer after the command.
- int OutputFileFormat (char ∗yytext)

  The OutputFileFormat function returns an integer that represents the format of the output file of the application.
- int OutputRisingFalling (char ∗yytext)

  The OutputRisingFalling function returns an integer that decides whether the trigger occurs on the rising or falling edge of the signal.
- int OutputNIMTTL (char ∗yytext)

  The OutputNIMTTL function returns an integer that represents the type of the front panel I/O LEMO connectors.
- int YesNoAnswer (char ∗yytext)

  The YesNoAnswer function returns an integer that represents yes or no.
- int GetAcquisitionMode (char ∗yytext)

  The GetAcquisitionMode function returns an integer that represents the acquisition mode.
- void GetOpenInformation (char ∗yytext, ConfObject ∗mioconfig)

  The GetOpenInformation function gets the information for opening the digitizer from the yytext string and puts it in a ConfObject.
- void GetWriteRegisterInformation (char ∗yytext, ConfObject ∗mioconfig)

  The GetWriteRegisterInformation function gets the information for writing a specific register of the digitizer from the yytext string and puts it in a ConfObject.
- void ChInformation (char ∗yytext, ConfObject ∗mioconfig)

  The ChInformation function gets the information for setting a specific channel of the digitizer from the yytext string and puts it in a ConfObject.
- void GroupInformation (char ∗yytext, ConfObject ∗mioconfig)

  The GroupInformation function gets the information for setting a specific group of the digitizer from the yytext string and puts it in a ConfObject.
- void AllInformation (char ∗yytext, ConfObject ∗mioconfig)

  The AllInformation function gets the information for setting all the channels of the digitizer from the yytext string and puts it in a ConfObject.
- void FastInformation (char ∗yytext, ConfObject ∗mioconfig)

  The FastInformation function gets the information for setting TRn channels of the digitizer from the yytext string and puts it in a ConfObject.
- void ChannelEnableMask (char ∗yytext, ConfObject ∗mioconfig)

  The ChannelEnableMask function gets the information for enabling or not the input from the channels.
- void GroupEnableMask (char ∗yytext, ConfObject ∗mioconfig)

  The GroupEnableMask function gets the information for enabling or not the input from the groups.
- void ChannelTriggerEnableMask (char ∗yytext, ConfObject ∗mioconfig)

  The ChannelTriggerEnableMask function gets the information for setting TRn channels of the digitizer from the yytext string and puts it in a ConfObject.

6.5.1 Detailed Description

This file contains the declaration of the utility functions used by the scanner. These functions are called in the flex file.

They help the yylex() function to extract the information from the matched strings.

Author

Daniele Berto

Definition in file AnalizzatoreUtils.h.

6.5.2 Function Documentation

6.5.2.1 void AllInformation (char ∗ *yytext,* ConfObject ∗ *mioconfig*)

The AllInformation function gets the information for setting all the channels of the digitizer from the yytext string and puts it in a ConfObject.

Ex. yytext contains the string "ALL TRIGGER_THRESHOLD 0x10000000", so we can call AllInformation (yytext, mioconfig).

Parameters

yytext	contains a matched string.
mioconfig	is where the retrieved informations are stored.

Returns

void

Definition at line 347 of file AnalizzatoreUtils.c.

6.5.2.2 void ChannelEnableMask (char ∗ *yytext,* ConfObject ∗ *mioconfig*)

The ChannelEnableMask function gets the information for enabling or not the input from the channels.

It gets them from the yytext string and puts it in a ConfObject.

Ex. yytext contains the string "CHANNEL_ENABLE_MASK 0x00000111", so we can call ChannelEnableMask (yytext, mioconfig).

Parameters

yytext	contains a matched string.
mioconfig	is where the retrieved informations are stored.

Returns

void

Definition at line 447 of file AnalizzatoreUtils.c.

6.5.2.3 void ChannelTriggerEnableMask (char ∗ *yytext,* ConfObject ∗ *mioconfig*)

The ChannelTriggerEnableMask function gets the information for setting TRn channels of the digitizer from the yytext string and puts it in a ConfObject.

Ex. yytext contains the string "FAST TRIGGER_THRESHOLD 0x10000000", so we can call ChannelTriggerEnable-Mask (yytext, mioconfig).

Parameters

yytext	contains a matched string.
mioconfig	is where the retrieved informations are stored.

Returns

void

Definition at line 499 of file AnalizzatoreUtils.c.

6.5.2.4 void ChInformation (char ∗ *yytext,* ConfObject ∗ *mioconfig*)

The ChInformation function gets the information for setting a specific channel of the digitizer from the yytext string and puts it in a ConfObject.

Ex. yytext contains the string "CH 1 CHANNEL_TRIGGER DISABLED", so we can call ChInformation (yytext, mioconfig).

Parameters

yytext	contains a matched string.
mioconfig	is where the retrieved informations are stored.

Returns

void

Definition at line 183 of file AnalizzatoreUtils.c.

6.5.2.5 void FastInformation (char ∗ *yytext,* ConfObject ∗ *mioconfig*)

The FastInformation function gets the information for setting TRn channels of the digitizer from the yytext string and puts it in a ConfObject.

Ex. yytext contains the string "FAST TRIGGER_THRESHOLD 0x10000000", so we can call FastInformation (yytext, mioconfig).

Parameters

yytext	contains a matched string.
mioconfig	is where the retrieved informations are stored.

Returns

void

Definition at line 401 of file AnalizzatoreUtils.c.

6.5.2.6 int FindIntegerValue (char ∗ *yytext*)

The FindIntegerValue function returns the value of the first integer after the command.

Ex: the string "POST_TRIGGER 20" is in yytext and we call value = find_integer_value(yytext). After that value contains contains the integer 20.

Parameters

yytext	contains a matched string.

Returns

int

Definition at line 51 of file AnalizzatoreUtils.c.

6.5.2.7 char∗ FindPointer (char ∗ *yytext*)

The FindPointer function returns a pointer to the first useful character after the command.

Ex: the string "POST_TRIGGER 20" is in yytext and we call punt = find_pointer(yytext). After that punt contains a pointer to the character '2'. This function is used by many other functions of this file.

Parameters

yytext	contains a matched string.

Returns

char

Definition at line 38 of file AnalizzatoreUtils.c.

6.5.2.8 int GetAcquisitionMode (char ∗ *yytext*)

The GetAcquisitionMode function returns an integer that represents the acquisition mode.

Values: 0 = ACQUISITION_ONLY, 1 = ACQUISITION_AND_TRGOUT, 2 = DISABLED, other=UNSET.

Parameters

yytext	contains a matched string.

Returns

int

Definition at line 113 of file AnalizzatoreUtils.c.

6.5.2.9 void GetOpenInformation (char ∗ *yytext*, ConfObject ∗ *mioconfig*)

The GetOpenInformation function gets the information for opening the digitizer from the yytext string and puts it in a ConfObject.

Ex. yytext contains the string "OPEN PCI 0 0 0x11110000", so we can call GetOpenInformation (yytext, mioconfig).

Parameters

yytext	contains a matched string.
mioconfig	is where the retrieved informations are stored.

Returns

void

Definition at line 132 of file AnalizzatoreUtils.c.

6.5.2.10 void GetWriteRegisterInformation (char ∗ *yytext,* ConfObject ∗ *mioconfig*)

The GetWriteRegisterInformation function gets the information for writing a specific register of the digitizer from the yytext string and puts it in a ConfObject.

Ex. yytext contains the string "WRITE_REGISTER 1080 0000 0100", so we can call GetWriteRegisterInformation (yytext, mioconfig).

Parameters

yytext	contains a matched string.
mioconfig	is where the retrieved informations are stored.

Returns

void

Definition at line 162 of file AnalizzatoreUtils.c.

6.5.2.11 void GroupEnableMask (char ∗ *yytext,* ConfObject ∗ *mioconfig*)

The GroupEnableMask function gets the information for enabling or not the input from the groups.

It gets them from the yytext string and puts it in a ConfObject.

Ex. yytext contains the string "GROUP_ENABLE_MASK 0x10000000", so we can call GroupEnableMask (yytext, mioconfig).

Parameters

yytext	contains a matched string.
mioconfig	is where the retrieved informations are stored.

Returns

void

Definition at line 473 of file AnalizzatoreUtils.c.

6.5.2.12 void GroupInformation (char ∗ *yytext,* ConfObject ∗ *mioconfig*)

The GroupInformation function gets the information for setting a specific group of the digitizer from the yytext string and puts it in a ConfObject.

Ex. yytext contains the string "GR 1 CHANNEL_TRIGGER DISABLED", so we can call GroupInformation (yytext, mioconfig).

Parameters

yytext	contains a matched string.
mioconfig	is where the retrieved informations are stored.

Returns

void

Definition at line 277 of file AnalizzatoreUtils.c.

6.5.2.13 int OutputFileFormat (char ∗ *yytext*)

The OutputFileFormat function returns an integer that represents the format of the output file of the application.

See ConfObject documentation for other informations.

Parameters

yytext	contains a matched string.

Returns

int

Definition at line 61 of file AnalizzatoreUtils.c.

6.5.2.14 int OutputNIMTTL (char ∗ *yytext*)

The OutputNIMTTL function returns an integer that represents the type of the front panel I/O LEMO connectors.

See ConfObject documentation for other informations.

Parameters

yytext	contains a matched string.

Returns

int

Definition at line 87 of file AnalizzatoreUtils.c.

6.5.2.15 int OutputRisingFalling (char ∗ *yytext*)

The OutputRisingFalling function returns an integer that decides whether the trigger occurs on the rising or falling edge of the signal.

See ConfObject documentation for other informations.

Parameters

yytext	contains a matched string.

Returns

int

Definition at line 74 of file AnalizzatoreUtils.c.

6.5.2.16 int YesNoAnswer (char ∗ *yytext*)

The YesNoAnswer function returns an integer that represents yes or no.

Values: 0 = NO, 1 = YES, other=UNSET.

Parameters

yytext	contains a matched string.

Returns

int

Definition at line 100 of file AnalizzatoreUtils.c.

6.6 AnalizzatoreUtils.h

```

#include <stdlib.h>
#include <string.h>
#include <assert.h>

char *FindPointer (char *yytext);

int FindIntegerValue (char *yytext);

int OutputFileFormat (char *yytext);

int OutputRisingFalling (char *yytext);

int OutputNIMTTL (char *yytext);

int YesNoAnswer (char *yytext);

int GetAcquisitionMode (char *yytext);

void GetOpenInformation (char *yytext, ConfObject * mioconfig);

void GetWriteRegisterInformation (char *yytext,
ConfObject * mioconfig);

void ChInformation (char *yytext, ConfObject * mioconfig);

void GroupInformation (char *yytext, ConfObject * mioconfig);

void AllInformation (char *yytext, ConfObject * mioconfig);

void FastInformation (char *yytext, ConfObject * mioconfig);

void ChannelEnableMask (char *yytext, ConfObject * mioconfig);

void GroupEnableMask (char *yytext, ConfObject * mioconfig);

void ChannelTriggerEnableMask (char *yytext, ConfObject * mioconfig);
```

6.7 ApplicationSetup.cpp File Reference

```
#include "ConfObject.h"
#include "ApplicationSetup.h"
#include "OutputModule.h"
#include "DefineGeneral.h"
#include <CAENDigitizer.h>
#include <stdio.h>
#include <unistd.h>
#include <ctype.h>
```

6.7.1 Detailed Description

Il singleton ApplicationSetup permette di salvare e rendere disponibili in tutte le parti del programma le informazioni necessarie per il funzionamento dell'applicazione. Esse sono inserite dall'utente lanciando il programma o inserendo l'apposito input ad applicazione avviata. Dunque, ApplicationSetup ricava le informazioni da argc e argv, cioe' dagli argomenti del main. Occorre che la funzione main chiami il metodo ApplicationSetupSet(argc, argv) per riempire ApplicationSetup. Gli argomenti del main vengono analizzati in modo classico, utilizzando la funzione "getopt".

Author

Daniele Berto

Definition in file ApplicationSetup.cpp.

6.8 ApplicationSetup.cpp

```

#include "ConfObject.h"
#include "ApplicationSetup.h"
#include "OutputModule.h"
#include "DefineGeneral.h"
#include <CAENDigitizer.h>
#include <stdio.h>
#include <unistd.h>
#include <ctype.h>

/*
Implementazione standard dei metodi per implementare il singleton design pattern
*/
ApplicationSetup *
  ApplicationSetup::application_setup_pInstance = NULL;

ApplicationSetup *
ApplicationSetup::Instance ()
{
  if (!application_setup_pInstance)    // Only allow one instance of class to be generated.
    application_setup_pInstance = new ApplicationSetup ();

  return application_setup_pInstance;
}


ApplicationSetup::ApplicationSetup ()
{
  channel_visualized = 0;

  imset = 0;

  application_setup_log_file_path = (const char *)malloc ((strlen ("
./LogFile") + 1));

  strcpy ((char *) application_setup_log_file_path,"./LogFile");

  application_setup_data_file_path =(const char *) malloc ((strlen ("
./RawData/data.txt") + 1));
  strcpy ((char *) application_setup_data_file_path, "./RawData/data.txt");

  application_setup_conf_file_path = (const char *)malloc ((strlen ("
./ConfigurationFile") + 1));
  strcpy ((char *) application_setup_conf_file_path, "./ConfigurationFile")
;

  application_setup_input_mode = (const char *) malloc ((strlen ("default") + 1
));
  strcpy ((char *) application_setup_input_mode, "default");

  application_setup_data_file_size_path = (const char *) malloc (
strlen (application_setup_data_file_path) + 3);
  strcpy ((char *) application_setup_data_file_size_path,
application_setup_data_file_path);
  strcat ((char *) application_setup_data_file_size_path, "sz");

  application_setup_data_file_punt = fopen (
application_setup_data_file_path, "a");
  application_setup_data_file_size_punt = fopen (
application_setup_data_file_size_path, "a");
}


void
ApplicationSetup::ApplicationSetupSet (int argc_arg, char **argv_arg)
{
  int i;
  argc = argc_arg;
  argv = (char **) malloc (argc * sizeof (char *));
  for (i = 0; i < argc; i++)
    {
      argv[i] = (char *) malloc (sizeof (char) * (strlen (argv_arg[i]) + 1));
      strcpy (argv[i], argv_arg[i]);
    }
  ApplicationSetup::ArgumentsParsing ();
  ApplicationSetup::FetchInputMode (
application_setup_input_mode);
}


int
ApplicationSetup:: ApplicationSetupDataFileModify (const
char *application_setup_data_file_path_arg)
{
  free ((void *) application_setup_data_file_path);
  application_setup_data_file_path = NULL;
  application_setup_data_file_path = (char *) malloc (strlen (application_setup_data_file_path_arg) + 1);
  strcpy ((char *) application_setup_data_file_path, application_setup_data_file_path_arg);

  if (application_setup_data_file_punt != NULL)
    fclose (application_setup_data_file_punt);

  application_setup_data_file_punt = fopen (application_setup_data_file_path, "a");

  free ((void *) application_setup_data_file_size_path);
  application_setup_data_file_size_path = NULL;
  application_setup_data_file_size_path = (char *) malloc (strlen (application_setup_data_file_path_arg) + 3);
  strcpy ((char *) application_setup_data_file_size_path, application_setup_data_file_path_arg);
  strcat ((char *) application_setup_data_file_size_path, "sz");

  if (application_setup_data_file_size_punt != NULL)
    fclose (application_setup_data_file_size_punt);

  application_setup_data_file_size_punt = fopen (application_setup_data_file_size_path, "a");
}


FILE *
ApplicationSetup::ApplicationSetupGetDataFilePunt ()
{
  return application_setup_data_file_punt;
}


FILE *
ApplicationSetup::ApplicationSetupGetDataFileSizePunt ()
{
  return application_setup_data_file_size_punt;
}


void
ApplicationSetup::FetchInputMode (const char *application_setup_input_mode)
{

  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  if (!strcmp ("user", application_setup_input_mode))
    {
      input_mode = 0;
      output_module->Output("User command mode activated\n");
    }

  else if (!strcmp ("tcp", application_setup_input_mode))
    {
      input_mode = 1;
      output_module->Output("Tcp command mode activated\n");
    }

  else if (!strcmp ("all", application_setup_input_mode))
    {
      input_mode = 2;
      output_module->Output("Tcp and User command mode activated\n");
    }

  else
    {
      output_module->Output("Tcp and User command mode activated\n");
      input_mode = 2;
    }
}


void
ApplicationSetup::ArgumentsParsing ()
{

  int c = 0;

  opterr = 0;

  while ((c = getopt (argc, argv, "f:m:d:l:")) != -1)
    switch (c)
```

```
      {
      case 'f':
    application_setup_conf_file_path =
      (char *) malloc (strlen (optarg) + 1);
    strcpy ((char *) application_setup_conf_file_path, optarg);
    break;
      case 'm':
    application_setup_input_mode = (char *) malloc (strlen (optarg) + 1);
    strcpy ((char *) application_setup_input_mode, optarg);
    break;
      case 'd':
    ApplicationSetup::ApplicationSetupDataFileModify (
optarg);
    break;
      case 'l':
    application_setup_log_file_path =
      (char *) malloc (strlen (optarg) + 1);
    strcpy ((char *) application_setup_log_file_path, optarg);
    break;
      case '?':
    if (optopt == 'f' || optopt == 'm' || optopt == 'd' || optopt == 'l')
      fprintf (stderr, "Option -%c requires an argument.\n", optopt);
    else if (isprint (optopt))
      fprintf (stderr, "Unknown option `-%c'.\n", optopt);
    else
      fprintf (stderr, "Unknown option character `\\x%x'.\n", optopt);
    break;
      }
}
```

6.9 ApplicationSetup.h File Reference

Classes

- class ApplicationSetup

  The ApplicationSetup class gets the application settings from the main parameters.

6.9.1 Detailed Description

Author

Daniele Berto

Definition in file ApplicationSetup.h.

6.10 ApplicationSetup.h

```

class ApplicationSetup
{
private:

  static ApplicationSetup *application_setup_pInstance;

    ApplicationSetup ();
public:

  int channel_visualized;

  static ApplicationSetup *Instance ();

  ConfObject application_setup_conf_object;

  FILE *application_setup_conf_file;

  int argc;

  int imset;

  char **argv;

  int ApplicationSetupDataFileModify (const char
                        *application_setup_data_file_path_arg);

  FILE *ApplicationSetupGetDataFilePunt ();

  FILE *ApplicationSetupGetDataFileSizePunt ();

  const char *application_setup_conf_file_path;

  const char *application_setup_log_file_path;

  const char *application_setup_data_file_path;

  const char *application_setup_data_file_size_path;

  const char *application_setup_input_mode;

  FILE *application_setup_data_file_punt;

  FILE *application_setup_data_file_size_punt;

  int input_mode;

  void ArgumentsParsing ();

  void ApplicationSetupSet (int argc, char **argv);

  void FetchInputMode (const char *application_setup_input_mode);
};
```

6.11 ClientApplication.c File Reference

This file contains the client_application.

```
#include "DefineGeneral.h"
#include "DefineCommands.h"
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <regex.h>
#include <stdbool.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <ctype.h>
```

Functions

- char ∗ FindPointer (char ∗yytext)

  The FindPointer function returns a pointer to the first useful character after the command.
- void ∗ ricevitore_function (void ∗nothing)
- bool reg_matches (const char ∗str, const char ∗pattern)

  Confronta la stringa puntata da str con il pattern indicato da pattern. Restituisci true se e' possibile trovare pattern in str, false al contrario.
- void Help ()
- int command_parser (char ∗inputline, char ∗sendline)

  The command_parser function returns a number which correspond to the command inserted by the user in accordance with the our conventions.
- int main (int argc, char ∗∗argv)

  The main function of the client application.

Variables

- int i
- int sockfd
- int n
- int comando_da_inviare
- char inputline [1000]
- char sendline [1000]
- char receiveline [1000]
- int go = 0
- int no_new_line = 0

6.11.1 Detailed Description

This file contains the client_application.

Definition in file ClientApplication.c.

6.11.2 Function Documentation

6.11.2.1 int command_parser (char ∗ *inputline,* char ∗ *sendline*)

The command_parser function returns a number which correspond to the command inserted by the user in accordance with the our conventions.

See DefineCommands.h file for informations about the codes of the available commands.

Parameters

inputline	contains the command being interpreted
sendline	contains the informations being sent via TCP/IP for composite commands.

Returns

int

Definition at line 142 of file ClientApplication.c.

6.11.2.2 char∗ FindPointer (char ∗ *yytext*)

The FindPointer function returns a pointer to the first useful character after the command.

Ex: the string "POST_TRIGGER 20" is in yytext and we call punt = find_pointer(yytext). After that punt contains a pointer to the character ’2’. This function is used by many other functions of this file.

Parameters

yytext	contains a matched string.

Returns

char

Definition at line 44 of file ClientApplication.c.

6.11.2.3 void Help ( )

Definition at line 102 of file ClientApplication.c.

6.11.2.4 int main (int *argc,* char ∗∗ *argv*)

The main function of the client application.

Parameters

argc	is the number of the main arguments
argv	are the main arguments

Returns

int

Definition at line 286 of file ClientApplication.c.

6.11.2.5 bool reg_matches (const char ∗ *str,* const char ∗ *pattern*)

Confronta la stringa puntata da str con il pattern indicato da pattern. Restituisci true se e' possibile trovare pattern in str, false al contrario.

Definition at line 84 of file ClientApplication.c.

6.11.2.6 void∗ ricevitore_function (void ∗ *nothing*)

Definition at line 57 of file ClientApplication.c.

6.11.3 Variable Documentation

6.11.3.1 int comando_da_inviare

Definition at line 35 of file ClientApplication.c.

6.11.3.2 int go = 0

Definition at line 39 of file ClientApplication.c.

6.11.3.3 int i

Definition at line 33 of file ClientApplication.c.

6.11.3.4 char inputline[1000]

Definition at line 36 of file ClientApplication.c.

6.11.3.5 int n

Definition at line 34 of file ClientApplication.c.

6.11.3.6 int no_new_line = 0

Definition at line 40 of file ClientApplication.c.

6.11.3.7 char receiveline[1000]

Definition at line 38 of file ClientApplication.c.

6.11.3.8 char sendline[1000]

Definition at line 37 of file ClientApplication.c.

6.11.3.9 int sockfd

Definition at line 34 of file ClientApplication.c.

6.12 ClientApplication.c

```
00001
00008 /*
00009 Questo sorgente, se compilato con il make file contenuto nella cartella principale del progetto, produce il
       programma objectdumpclient, il lato client del programma
00010 objectdump.
00011 Objectdumpclient consente di inviare ad objectdump comandi in input via TCP/IP. La lista dei codici che il
       client puo' inviare e' contenuta nel file DefineCommands.h.
00012 Ricordo che affinche' il server riconosca i comandi inviati via TCP/IP, i dati inviati devono essere un
       array di caratteri contenente nelle prime tre posizioni i numeri
00013 25 21 17 e poi il codice del comando da inviare.
00014 L'output mandato dal server e' ascoltato da un thread separato che esegue la funzione
       "ricevitore_function".
00015 Per lanciare il programma occorre aggiungere l'opzione -i serveraddress dopo il nome dell'eseguibile.
00016 */
00017
00018 #include "DefineGeneral.h"
00019 #include "DefineCommands.h"
00020 #include <sys/socket.h>
00021 #include <netinet/in.h>
00022 #include <arpa/inet.h>
00023 #include <stdio.h>
00024 #include <string.h>
00025 #include <regex.h>
00026 #include <stdbool.h>
00027 #include <stdlib.h>
00028 #include <pthread.h>
00029 #include <unistd.h>
00030 #include <ctype.h>
00031
00032 //Variabili globali contenenti la socket di comunicazione col server e i dati da inviare e ricevere.
00033   int i;
00034   int sockfd, n;
00035   int comando_da_inviare;
00036   char inputline[STANDARDBUFFERLIMIT];
00037   char sendline[STANDARDBUFFERLIMIT];
00038   char receiveline[STANDARDBUFFERLIMIT];
00039   int go = 0;
00040   int no_new_line = 0;
00041
00042 //Funzione che restituisce un puntatore alla prima parola contenuta in yytext
00043 char *
00044 FindPointer (char *yytext)
00045 {
00046   int i = 0;
00047   char *punt;
00048   while (yytext[i] != ' ' && yytext[i] != '\t')
00049     i++;
00050   while (yytext[i] == ' ' || yytext[i] == '\t')
00051     i++;
00052   punt = yytext + i;
00053   return punt;
00054 }
00055
00056 //Funzione eseguita dal thread che riceve l'output dal server
00057 void * ricevitore_function (void * nothing)
00058 {
00059 int i;
00060     while (go)
00061     {
00062             bzero(receiveline, STANDARDBUFFERLIMIT);
00063
00064             if (recv (sockfd, receiveline, STANDARDBUFFERLIMIT, 0) <= 0
)
00065                     {
00066                             fprintf (stdout, "Comunicazione col server interrotta\n");
00067                                 shutdown (sockfd, 2);
00068                                 go = 0;
00069                         } else {
00070                        //If necessario per stampare correttamente a video l'output del server.
```

```
                    if (receiveline[strlen(receiveline)-1] == 10)
                        fprintf(stdout, "%s", receiveline);
                    else if(receiveline[strlen(receiveline)-1] != 10)
                        {
                        fprintf(stdout, "%s", receiveline);
                            if (strlen(receiveline) > 1 && no_new_line == 0)
                                fprintf (stdout, "\n");
                        } // else if(receiveline[strlen(receiveline)-1] != 10)
                    } //} else {
    } //while (go)
} //void * ricevitore_function (void * nothing)

bool reg_matches(const char *str, const char *pattern)
{
    regex_t re;
    int ret;

    if (regcomp(&re, pattern, REG_EXTENDED) != 0)
        return false;

    ret = regexec(&re, str, (size_t) 0, NULL, 0);
    regfree(&re);

    if (ret == 0)
        return true;

    return false;
}

//Stampa i comandi disponibile. La funzione e' chiamata quando l'utente digita il comando help.
void Help()
{
  printf ("Available command list:\n");
  printf ("init: open the digitizer\n");
  printf ("setup: setup the digitizer\n");
  printf ("start: start the data acquisition\n");
  printf ("stop: stop the data acquisition\n");
  printf ("prestart: start the preprocessing thread\n");
  printf ("prestop: stop the preprocessing thread\n");
  printf ("vistart [channelnumber]: start the visualization thread\n");
  printf ("vistop: stop the visualization thread\n");
  printf ("rawstart: start the raw data writing thread\n");
  printf ("rawstop: stop the raw data writing thread\n");
  printf ("close: close the digitizer\n");
  printf ("send: send a software trigger\n");
  printf ("help\n");
  printf ("check: check the correctness of the configuration file\n");
  printf ("chkconf: print the content of the configuration file\n");
  printf ("write register 0x[register] 0x[data]\n");
  printf ("read register 0x[register]\n");
  printf ("-f [conf file path]: change the configuration file path\n");
  printf ("-d [data file path]: change the data file path\n");
  printf ("-l [log file path]: change the log file path\n");
  printf ("print: print the internal configuration object used to configure the digitizer\n");
  printf ("print files: print the path of the configuration file, of the data file and of the log file\n");
  printf ("status: print the status of the threads acquisition, preprocessing, raw data and visualization\n");
  printf ("more: display the content of the logfile\n");
  printf ("exit: quit program\n");
  printf ("quit: quit program\n");
}


int
command_parser (char *inputline , char *sendline)
{ char * my_punt;
  const char *my_punt_const;
  no_new_line = 0;

  if (reg_matches(inputline, "^[iI][Nn][Ii][Tt][ \t]*$"))
    {
      //printf ("1\n");
      return INIT;
    }
  else if (reg_matches (inputline, "^[Ss][Ee][Tt][Uu][Pp][ \t]*$"))
    {
      //printf ("2\n");
      return SETUP;
    }
  else if (reg_matches (inputline, "^[pP][rR][eE][sS][tT][aA][rR][tT][ \t]*$"))
    {
      //printf ("3\n");
      return PRESTART;
    }
  else if (reg_matches (inputline, "^[pP][Rr][Ee][Ss][Tt][Oo][Pp][ \t]*$"))
    {
      //printf ("4\n");
      return PRESTOP;
    }
  else if (reg_matches (inputline, "^[sS][tT][aA][rR][tT][ \t]*$"))
    {
      //printf ("7\n");
      return START;
    }
  else if (reg_matches (inputline, "^[sS][tT][oO][pP][ \t]*$"))
    {
      //printf ("8\n");
      return STOP;
    }
  else if (reg_matches (inputline, "^[Ss][Ee][Nn][Dd][ \t]*$"))
    {
      //printf ("9\n");
      return SEND;
    }
  else if (reg_matches (inputline, "^[cC][lL][oO][sS][eE][ \t]*$"))
    {
      //printf ("10\n");
      return CLOSE;
    }
  else if (reg_matches (inputline, "^[qQ][uU][iI][tT][ \t]*$"))
    {
      //printf ("11\n");
      return QUIT;
    }
  else if (reg_matches (inputline, "^[rR][aA][wW][sS][tT][aA][rR][tT][ \t]*$"))
    {
      //printf ("12\n");
      return RAWSTART;
    }
  else if (reg_matches (inputline, "^[Rr][Aa][Ww][Ss][Tt][Oo][Pp][ \t]*$"))
    {
      //printf ("13\n");
      return RAWSTOP;
    }
  else if (reg_matches (inputline, "^[pP][rR][iI][nN][tT][ \t]*$"))
    {
      //printf ("14\n");
      return PRINT;
    }
  else if (reg_matches (inputline, "^[Cc][Hh][Ee][Cc][Kk][ \t]*$"))
    {
      //printf ("16\n");
      return CHECK;
    }
  else if (reg_matches (inputline, "^[Cc][Hh][Kk][Cc][Oo][Nn][Ff][ \t]*$"))
    {
      //printf ("17\n");
      return CHKCONF;
    }
  else if (reg_matches (inputline, "^[Mm][Oo][Rr][Ee][ \t]*$"))
    {
      no_new_line = 1;
      //printf ("18\n");
      return MORE;
    }
  else if (reg_matches (inputline, "^[Ss][Tt][Aa][Tt][Uu][Ss][ \t]*$"))
    {
      //printf ("17\n");
      return STATUS;
    }
  else if (reg_matches (inputline, "^[-][Ff][ ].+$"))
    {
      //printf ("18\n");
      return CHANGECONF;
    }
  else if (reg_matches (inputline, "^[-][Dd][ ].+$"))
    {
      //printf ("18\n");
      return CHANGEDATA;
    }
  else if (reg_matches (inputline, "^[-][Ll][ ].+$"))
    {
      //printf ("18\n");
      return CHANGELOG;
    }
  else if (reg_matches (inputline, "^[pP][rR][iI][nN][tT][ \t]+[Ff][Ii][Ll][Ee][Ss][ \t]*$"))
    {
      //printf ("18\n");
      return PRINTFILES;
    }
  else if (reg_matches (inputline, "^[Ww][Rr][Ii][Tt][Ee][ \t]+[Rr][Ee][Gg][Ii][Ss][Tt][Ee][Rr][
 \t]+(0x[0-9a-fA-F]{1,16})[ \t]+(0x[0-9a-fA-F]{1,16})[ \t]*$"))  //5==print
        {
```

```
          my_punt = FindPointer (inputline);
          my_punt = FindPointer (my_punt);
              strcpy(sendline + 4, my_punt);
              return WRITEREGISTER;
        }
  else if (reg_matches (inputline, "^[Rr][Ee][Aa][Dd][ \t]+[Rr][Ee][Gg][Ii][Ss][Tt][Ee][Rr][ \t
]+(0x[0-9a-fA-F]{1,4})[ \t]*$"))  //5==print
        {
          my_punt = FindPointer (inputline);
          my_punt = FindPointer (my_punt);
              strcpy(sendline + 4, my_punt);
              return READREGISTER;
        }
  else if (reg_matches (inputline, "^[hH][eE][lL][pP][ \t]*$"))
    {
      Help();
      return -1;
    }
  else if (reg_matches (inputline, "^[Ee][Xx][Ii][Tt]$"))
    {
      fprintf(stdout, "Exiting...\n");
      return -1;
    }
  else
    {
      fprintf (stderr, "Unrecognized command\n");
      return -1;
    }
}

int
main (int argc, char **argv)
{

//Variabili per ottenere l'indirizzo del server dagli argomenti del main.
  const char *server_address;
  server_address == NULL;

  int c = 0;

  int flag_arg = 0;

  opterr = 0;

  while ((c = getopt (argc, argv, "i:")) != -1)
    switch (c)
      {
      case 'i':
    server_address = (char *)malloc(strlen(optarg) + 1);
        strcpy((char *)server_address, optarg);
    flag_arg = 1;
        break;
      case '?':
        if (optopt == 'i')
          fprintf (stderr, "Option -%c requires an argument.\n", optopt);
        else if (isprint (optopt))
          fprintf (stderr, "Unknown option `-%c'.\n", optopt);
        else
          fprintf (stderr,
                   "Unknown option character `\\x%x'.\n",
                   optopt);
        break;
      }

  if (flag_arg == 0)
    {
    fprintf(stderr, "You have not insert server address: use -i flag\n");
    fprintf(stderr, "usage: [executablepath] -i [serveraddress]\n");
    return 1;
    }

//Inserisco queste informazioni dentro sendline in modo tale che il server riconosca i dati inviategli.
  sendline[0] = 25;
  sendline[1] = 21;
  sendline[2] = 17;
  sendline[4] = '\0';

  const char * my_punt;

//thread id del thread che ascolta i messaggi provenienti dal server
  pthread_t ricevitore;

//Codice necessario all'apertura di una socket con il client.
  struct sockaddr_in servaddr, cliaddr;
  char recvline[STANDARDBUFFERLIMIT];
  sockfd = socket (AF_INET, SOCK_STREAM, 0);
  bzero (&servaddr, sizeof (servaddr));
  servaddr.sin_family = AF_INET;
  servaddr.sin_addr.s_addr = inet_addr (server_address);
  servaddr.sin_port = htons (1111);

  fprintf(stdout, "Welcome to objectDump tcp service, press help for getting the available command list\n")
;

  connect (sockfd, (struct sockaddr *) &servaddr, sizeof (servaddr));
  perror(" ");

//Creo il thread ricevitore, controllato dalla variabile go.
  go = 1;
  pthread_create(&ricevitore, NULL, ricevitore_function, NULL);

//Ciclo di fetching dell'input da tastiera il ciclo termina quando l'utente inserisce il comando exit.
  do
    {
      fflush(stdout);

      //Prelevo un input da tastiera
      fgets (inputline, STANDARDBUFFERLIMIT, stdin);
      if (inputline[strlen (inputline) - 1] == '\n')
            inputline[strlen (inputline) - 1] = '\0';

      //Metto in sendline il codice corrispondente al comando inserito dall'utente.
      //Se il comando non e' riconosciuto, viene restituito -1 e non e' inviato nulla al server.
      sendline[3] = (char) command_parser (inputline,
sendline);

      //Se il client invia un comando per modificare il path dei files usati dal server, e' necessario
 ricavare dalla stringa in input il nuovo path
      //e inviarlo al server.
      if (sendline[3] == CHANGECONF || sendline[3] ==
CHANGEDATA || sendline[3] == CHANGELOG)
            {my_punt = inputline + 3;
        strcpy(sendline + 4, my_punt);
        }

      //Se il comando inserito dall'utente non e' stato riconosciuto, al server non e' inviato nella.
      if (sendline[3] != -1)
        {
        send (sockfd, sendline, STANDARDBUFFERLIMIT, 0);
        }
    }
  //Il ciclo termina quando l'utente inserisce il comando exit.
  while (!reg_matches (inputline, "^[Ee][Xx][Ii][Tt][ \t]*$") != 0);

  //Il programma deve terminare, quindi interrompo anche il thread ricevitore
  go = 0;

  //Chiudo la socket di comunicazione col server.
  shutdown (sockfd, 2);
} //int main (int argc, char **argv)


```

6.13 CommunicationObject.cpp File Reference

```
#include "DefineGeneral.h"
#include "DefineCommands.h"
#include "TcpUser.h"
#include "ConfObject.h"
#include "DigitizerErrorObject.h"
#include "LogFile.h"
#include "DigitizerObject.h"
#include "DigitizerObjectGeneric.h"
#include "RawData.h"
#include "DigitizerStateMachine.h"
#include "ApplicationSetup.h"
#include "Input.h"
#include "CommunicationObject.h"
#include "DigitizerFlowControl.h"
#include "ConfigurationConsistence.h"
#include "AnalizzatoreUtils.h"
#include "Analizzatore.h"
#include "OutputModule.h"
#include <pthread.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <thread>
#include <mutex>
#include <condition_variable>
```

6.13.1 Detailed Description

Author

Daniele Berto

Definition in file CommunicationObject.cpp.

6.14 CommunicationObject.cpp

```

#include "DefineGeneral.h"
#include "DefineCommands.h"
#include "TcpUser.h"
#include "ConfObject.h"
#include "DigitizerErrorObject.h"
#include "LogFile.h"
#include "DigitizerObject.h"
#include "DigitizerObjectGeneric.h"
#include "RawData.h"
#include "DigitizerStateMachine.h"
#include "ApplicationSetup.h"
#include "Input.h"
#include "CommunicationObject.h"
#include "DigitizerFlowControl.h"
#include "ConfigurationConsistence.h"
#include "AnalizzatoreUtils.h"
#include "Analizzatore.h"
#include "OutputModule.h"
#include <pthread.h>
#include <stdio.h>
```

```
#include<string.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<unistd.h>
#include <thread>
#include <mutex>
#include <condition_variable>

using namespace std;

void
CommunicationObject::Main ()
{

  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  int client_sock;  /*Socket descriptor del client. */
  int c;  /*Lunghezza di sockaddr_in. */
  struct sockaddr_in server, client;
  int socket_desc;
  socket_desc = socket (AF_INET, SOCK_STREAM, 0);

  /*Inizializzo la struttura sockaddr_in */
  server.sin_family = AF_INET;
  server.sin_addr.s_addr = INADDR_ANY;
  server.sin_port = htons (1111);

  /*Effettuo il bind */
  bind (socket_desc, (struct sockaddr *) &server, sizeof (server));

  while (go)
    {

      listen (socket_desc, 1);

      c = sizeof (struct sockaddr_in);

      if (go == 0)
    {
      break;
    }

      client_sock =
    accept (socket_desc, (struct sockaddr *) &client, (socklen_t *) & c);

      if (go == 0)
    {
      break;
    }

      if (client_sock >= 0)
    {
      output_module->OutputModuleStdoutOff();
      output_module->OutputModuleSockidOn(client_sock);
      output_module->TcpUserArrayInsert (client_sock);
      output_module->Output("Comunicazione col server stabilita\n");
      worker_thread = new thread (&CommunicationObject::Worker, this,
              (void *) &client_sock);
    }

    }                // while go

}


void
CommunicationObject::Worker (void *socket_desc)
{
  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  int j;
  int sock = *(int *) socket_desc;
  int read_size;
  char buffer[STANDARDBUFFERLIMIT];
  const char *my_punt;
  bzero (buffer, STANDARDBUFFERLIMIT);

  while (go)
    {
      bzero (buffer, STANDARDBUFFERLIMIT);
      if ((read_size = recv (sock, buffer, STANDARDBUFFERLIMIT, 0)) <= 0)
    {
      //fprintf (stderr, "Comunicazione con il client %d interrotta\nobjectDump>",
      //   sock);
      output_module->TcpUserArrayDelete (sock);
    }

      unique_lock<mutex> ReservedKeyBoardInputAreaHandle(ReservedKeyBoardInputArea);

      if (num_mex == MAXCOMMAND)
        BlockedProducerInput.wait(ReservedKeyBoardInputAreaHandle);
      coda = (coda + 1) % MAXCOMMAND;
      command[coda].command_sent_by_user = buffer[3];

      if (buffer[3] == CHANGECONF || buffer[3] == CHANGEDATA
      || buffer[3] == CHANGELOG || buffer[3] == WRITEREGISTER
      || buffer[3] == READREGISTER)
    {
      bzero (command[coda].first_parameter, STANDARDBUFFERLIMIT);
      my_punt = buffer + 4;
      strncpy (command[coda].first_parameter, my_punt, 999);
      fprintf (stderr, "command[coda].first_parameter: %s\n",
           command[coda].first_parameter);
    }

      command[coda].user_sockid = sock;
      num_mex++;
      pthread_mutex_lock (&DigitizerFlowControl::input_flow_mutex);
      pthread_cond_signal (&DigitizerFlowControl::input_flow_cond);
      pthread_mutex_unlock (&DigitizerFlowControl::input_flow_mutex);

      BlockedConsumerInput.notify_one();

    }               // while go

      BlockedConsumerInput.notify_one();

}


void
CommunicationObject::CommunicationObjectInit ()
{

  go = 1;

  main_thread = new thread (&CommunicationObject::Main, this);

}


TcpUser CommunicationObject::GetCommand ()
{
  TcpUser
    tmp;
  tmp.command_sent_by_user = 0;
  tmp.user_sockid = -1;
      unique_lock<mutex> ReservedKeyBoardInputAreaHandle(ReservedKeyBoardInputArea);

  if (num_mex == 0)
    {
      tmp.command_sent_by_user = 0;
      tmp.user_sockid = -1;
    }
  else
    {
      tmp = command[testa];

      command[testa].command_sent_by_user = 0;
      command[testa].user_sockid = -1;
      bzero (command[testa].first_parameter, STANDARDBUFFERLIMIT);

      testa = (testa + 1) % MAXCOMMAND;

      num_mex--;
      BlockedProducerInput.notify_one();
    }
  return tmp;
}


void
CommunicationObject::Finish ()
{
  int i;

  go = 0;

  for (i = 0; i < 10; i++)
      BlockedProducerInput.notify_one();

```

```
}


CommunicationObject::CommunicationObject ()
{
  num_mex = 0;
  coda = 0;
  testa = 1;
}
```

6.15 CommunicationObject.h File Reference

```
#include "DefineGeneral.h"
#include <pthread.h>
#include <thread>
#include <mutex>
#include <condition_variable>
```

Classes

- class CommunicationObject

6.16 CommunicationObject.h

```

#include "DefineGeneral.h"
#include <pthread.h>
#include <thread>
#include <mutex>
#include <condition_variable>

using namespace std;


class CommunicationObject
{
public:

  int num_mex;

  int testa;

  int coda;

  int go;

  void GetCommand (int *socketid, int *command);

  TcpUser GetCommand ();

  void CommunicationObjectInit ();

  void Main ();

  void Worker (void *socket_desc);

  TcpUser command[MAXCOMMAND];

  void Finish ();

  thread * main_thread;

  thread * worker_thread;

  mutex ReservedKeyBoardInputArea;

  condition_variable BlockedProducerInput;

  condition_variable BlockedConsumerInput;

  mutex Acquisition_Mutex;

  condition_variable Acquisition_Cond1;

  condition_variable Acquisition_Cond2;

    CommunicationObject ();
};
```

6.17 ConfigurationConsistence.cpp File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <CAENDigitizer.h>
#include "DefineGeneral.h"
#include "ConfObject.h"
#include "ApplicationSetup.h"
#include "Analizzatore.h"
#include "ConfigurationConsistence.h"
#include "OutputModule.h"
```

6.17.1 Detailed Description

Author

Daniele Berto

Definition in file ConfigurationConsistence.cpp.

6.18 ConfigurationConsistence.cpp

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <CAENDigitizer.h>
#include "DefineGeneral.h"
#include "ConfObject.h"
#include "ApplicationSetup.h"
#include "Analizzatore.h"
#include "ConfigurationConsistence.h"
#include "OutputModule.h"



int
ConfigurationConsistence::
ConfigurationConsistenceConfFileInit (const char *conf_file)
{

  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  int ret_flag = 0;
  ConfObject conf_object_private;


  if (AnalizzaInitPrint (&conf_object_private, conf_file))
    {
      fprintf (stderr, "%s: Errore: il file di configurazione non esiste\n",
           __func__);
      ret_flag = 1;
    output_module->Output("Errore: il file di configurazione non esiste\n");
      return ret_flag;
    }

  if (conf_object_private.LinkType == -1)
    {
      fprintf (stderr,
          "%s: Error: you have not specify LinkType in OPEN parameter\n",
          __func__);
      ret_flag++;
    output_module->Output("Errore: il file di configurazione non esiste\n");
    }

  if (conf_object_private.LinkNumber == -1)
    {
      fprintf (stderr,
          "%s: Error: you have not specify LinkNumber in OPEN parameter\n",
          __func__);
      ret_flag++;
      output_module->Output("Error: you have not specify LinkNumber in OPEN parameter\n");
    }

  if (conf_object_private.ConetNode == -1)
    {
      fprintf (stderr,
          "%s: Error: you have not specify ConetNode in OPEN parameter\n",
          __func__);
      ret_flag++;
      output_module->Output("Error: you have not specify ConetNode in OPEN parameter\n");
    }

  if (conf_object_private.VMEBaseAddress == -1)
    {
      fprintf (stderr,
          "%s: Error: you have not specify VMEBaseAddress in OPEN parameter\n",
          __func__);
      ret_flag++;
      output_module->Output("Error: you have not specify VMEBaseAddress in OPEN parameter\n");
    }


  if (ret_flag == 4)
    {
      fprintf (stderr,
          "%s: Ops, have you forgotten to insert OPEN attribute in the configuration file?\n",
          __func__);
      output_module->Output("Ops, have you forgotten to insert OPEN attribute in the configuration
 file?\n");
    }

  return ret_flag;

}  //int ConfigurationConsistence::ConfigurationConsistenceConfFileInit (const char * conf_file)



int
ConfigurationConsistence::
ConfigurationConsistenceConfFileInitNoPrint (const char *
conf_file)
{

  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  int ret_flag = 0;
  ConfObject conf_object_private;


  if (AnalizzaInit (&conf_object_private, conf_file))
    {
      ret_flag = 1;
     output_module->Output("Errore: il file di configurazione non esiste\n");
      return ret_flag;
    }

  if (conf_object_private.LinkType == -1)
    {
      ret_flag++;
      output_module->Output("Error: you have not specify LinkType in OPEN parameter\n");
    }

  if (conf_object_private.LinkNumber == -1)
    {
      ret_flag++;
      output_module->Output("Error: you have not specify LinkNumber in OPEN parameter\n");
    }

  if (conf_object_private.ConetNode == -1)
    {
      ret_flag++;
      output_module->Output("Error: you have not specify ConetNode in OPEN parameter\n");
    }

  if (conf_object_private.VMEBaseAddress == -1)
    {
      ret_flag++;
      output_module->Output("Error: you have not specify VMEBaseAddress in OPEN parameter\n");
    }


  if (ret_flag == 4)
    {
      output_module->Output("Ops, have you forgotten to insert OPEN attribute in the configuration
 file?\n");
    }

  return ret_flag;

}



int
ConfigurationConsistence::ConfigurationConsistenceConfFileSetupEssentialWithInitCheck
 (const char *conf_file)
{
  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  int ret_flag = 0;
  ConfObject conf_object_private;
  ret_flag =
    ConfigurationConsistence::
    ConfigurationConsistenceConfFileInit (conf_file);

  if (AnalizzaSetupPrint (&conf_object_private, conf_file))
    {
      ret_flag++;
      output_module->Output("Errore: il file di configurazione non esiste\n");
      return ret_flag;
    }

  if (conf_object_private.record_length == -1)
    {
      ret_flag++;
      output_module->Output("Error: you have not specify RECORD_LENGTH\n");
    }

  if (conf_object_private.max_num_events_BLT == -1)
    {
      ret_flag++;
      output_module->Output("Error: you have not specify MAX_NUM_EVENTS_BLT\n");
    }

  return ret_flag;

}



int
ConfigurationConsistence:: ConfigurationConsistenceConfFileSetupEssentialWithInitCheckNoPrint
 (const char *conf_file)
{
  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  int ret_flag = 0;
  ConfObject conf_object_private;
  ret_flag =
    ConfigurationConsistence::
    ConfigurationConsistenceConfFileInitNoPrint (conf_file);

  if (AnalizzaSetup (&conf_object_private, conf_file))
    {
      ret_flag++;
      output_module->Output("Errore: il file di configurazione non esiste\n");
      return ret_flag;
    }

  if (conf_object_private.record_length == -1)
    {
      ret_flag++;
      output_module->Output("Error: you have not specify RECORD_LENGTH\n");
    }

  if (conf_object_private.max_num_events_BLT == -1)
    {
      ret_flag++;
      output_module->Output("Error: you have not specify MAX_NUM_EVENTS_BLT\n");
    }

  return ret_flag;

}


int
ConfigurationConsistence::ConfigurationConsistenceConfFileSetupEssentialWithoutInitCheck
 (const char *conf_file)
{
  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  int ret_flag = 0;
  ConfObject conf_object_private;

  if (AnalizzaSetupPrint (&conf_object_private, conf_file))
    {
      ret_flag++;
      output_module->Output("Errore: il file di configurazione non esiste\n");
      return ret_flag;
    }

  if (conf_object_private.record_length == -1)
    {
      ret_flag++;
      output_module->Output("Error: you have not specify RECORD_LENGTH\n");
    }

  if (conf_object_private.max_num_events_BLT == -1)
    {
      ret_flag++;
      output_module->Output("Error: you have not specify MAX_NUM_EVENTS_BLT\n");
    }

  return ret_flag;

}



int
ConfigurationConsistence:: ConfigurationConsistenceConfFileSetupEssentialWithoutInitCheckNoPrint
 (const char *conf_file)
{
  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  int ret_flag = 0;
  ConfObject conf_object_private;

  if (AnalizzaSetup (&conf_object_private, conf_file))
    {
      ret_flag++;
      output_module->Output("Errore: il file di configurazione non esiste\n");
      return ret_flag;
    }

  if (conf_object_private.record_length == -1)
    {
      ret_flag++;
      output_module->Output("Error: you have not specify RECORD_LENGTH\n");
    }

  if (conf_object_private.max_num_events_BLT == -1)
    {
      ret_flag++;
      output_module->Output("Error: you have not specify MAX_NUM_EVENTS_BLT\n");
    }

  return ret_flag;

}
```

6.19 ConfigurationConsistence.h File Reference

Classes

- class ConfigurationConsistence

  The ConfigurationConsistence class provides useful methods for checking the consistence of a configuration file.

6.20 ConfigurationConsistence.h

```
class ConfigurationConsistence
{
public:

  int ConfigurationConsistenceConfFileInit (const char *conf_file_path)
;

  int ConfigurationConsistenceConfFileInitNoPrint (const char
                           *conf_file_path);

  int ConfigurationConsistenceConfFileSetupEssentialWithInitCheck
 (const char
                                   *conf_file_path);

  int
    ConfigurationConsistenceConfFileSetupEssentialWithInitCheckNoPrint
 (const
                                    char
                                    *conf_file_path);

  int ConfigurationConsistenceConfFileSetupEssentialWithoutInitCheck
 (const
                                      char
                                      *conf_file_path);

  int
    ConfigurationConsistenceConfFileSetupEssentialWithoutInitCheckNoPrint
    (const char *conf_file_path);
};
```

6.21 ConfObject.cpp File Reference

```
#include "DefineGeneral.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "ConfObject.h"
#include "ApplicationSetup.h"
#include "OutputModule.h"
```

6.22 ConfObject.cpp

```
#include "DefineGeneral.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "ConfObject.h"
#include "ApplicationSetup.h"
#include "OutputModule.h"


ConfObject::ConfObject ()
{
  LinkType = -1;
  LinkNumber = -1;
  ConetNode = -1;
  VMEBaseAddress = -1;
  DSR4_Frequency = -1;
  output_file_format = -1;
  gnuplot = NULL;
  header_yes_no = -1;
  record_length = -1;
  test_pattern = -1;
  desmod = -1;
  external_trigger_acquisition_mode = -1;
  fast_trigger_acquisition_mode = -1;
  enable_fast_trigger_digitizing = -1;
  max_num_events_BLT = -1;
  decimation_factor = -1;
  post_trigger = -1;
  rising_falling = -1;
  use_interrupt = -1;
  nim_ttl = -1;

  Address_register = -1;
  Mask_register = -1;
  Data_register = -1;

  enable_input = -1;
  dc_offset = -1;
  trigger_threshold = -1;

  channel_enable_mask = -1;
  group_enable_mask = -1;
  self_trigger_enable_mask = -1;
  self_trigger_enable_mask_mode = -1;

  //It could be useful to set a default value for the gnuplot path
  gnuplot = (char *)malloc(strlen("gnuplot") +1);
  strcpy (gnuplot, "gnuplot");
}


ChannelObject::ChannelObject ()
{
  set = -1;
  numChannel = -1;
  enable_input = -1;
  dc_offset = -1;
  trigger_threshold = -1;
  channel_trigger = -1;
}


GroupObject::GroupObject ()
{
  set = -1;
  numGroup = -1;
  enable_input = -1;
  dc_offset = -1;
  trigger_threshold = -1;
  group_trg_enable_mask = -1;
}


FastObject::FastObject ()
{
  set = -1;
  numFast = -1;
  dc_offset = -1;
  trigger_threshold = -1;
}


void
ChannelObject::PrintChannel ()
{
  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  char stringa[STANDARDBUFFERLIMIT];
  bzero (stringa, STANDARDBUFFERLIMIT);

  snprintf (stringa, STANDARDBUFFERLIMIT,
        "numchannel\t\t\t%d\nset\t\t\t\t%d\nenable_input\t\t\t%d\ndc_offset\t\t\t%d\ntrigger_threshold\t\t%d\nchannel_trigger\t\t\t%d\n\n",
        numChannel, set, enable_input, dc_offset, trigger_threshold,
        channel_trigger);

  output_module->Output(stringa);
}


void
GroupObject::PrintGroup ()
```

```
{
  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  char stringa[STANDARDBUFFERLIMIT];
  bzero (stringa, STANDARDBUFFERLIMIT);

  snprintf (stringa, STANDARDBUFFERLIMIT,
        "numgroup\t\t\t%d\nset\t\t\t\t%d\nenable_input\t\t\t%d\ndc_offset\t\t\t%d\ntrigger_threshold\t\t%d\ngroup_trg_enable_mask\t\t%d\n\n",
        numGroup, set, enable_input, dc_offset, trigger_threshold,
        group_trg_enable_mask);

  output_module->Output(stringa);
}


void
FastObject::PrintFast ()
{
  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  char stringa[STANDARDBUFFERLIMIT];
  bzero (stringa, STANDARDBUFFERLIMIT);

  snprintf (stringa, STANDARDBUFFERLIMIT,
        "numFast\t\t\t\t%d\nset\t\t\t\t%d\ndc_offset\t\t\t%d\ntrigger_threshold\t\t%d\n\n",
        numFast, set, dc_offset, trigger_threshold);

  output_module->Output(stringa);
}


void
ConfObject::PrintAllHuman ()
{

  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  char stringa[STANDARDBUFFERLIMIT];
  bzero (stringa, STANDARDBUFFERLIMIT);

  int i;

  const char *output_file_format_string;
  if (output_file_format == 0)
    output_file_format_string = "BINARY";
  else if (output_file_format == 1)
    output_file_format_string = "ASCII";
  else
    output_file_format_string = "UNSET";

  const char *header_yes_no_string;
  if (header_yes_no == 1)
    header_yes_no_string = "YES";
  else if (header_yes_no == 0)
    header_yes_no_string = "NO";
  else
    header_yes_no_string = "UNSET";

  const char *des_mod_string;
  if (desmod == 1)
    des_mod_string = "YES";
  else if (desmod == 0)
    des_mod_string = "NO";
  else
    des_mod_string = "UNSET";

  const char *external_trigger_acquisition_mode_string;
  if (external_trigger_acquisition_mode == 0)
    external_trigger_acquisition_mode_string = "ACQUISITION_ONLY";
  else if (external_trigger_acquisition_mode == 1)
    external_trigger_acquisition_mode_string = "ACQUISITION_AND_TRGOUT";
  else if (external_trigger_acquisition_mode == 2)
    external_trigger_acquisition_mode_string = "DISABLED";
  else
    external_trigger_acquisition_mode_string = "UNSET";

  const char *fast_trigger_acquisition_mode_string;
  if (fast_trigger_acquisition_mode == 0)
    fast_trigger_acquisition_mode_string = "ACQUISITION_ONLY";
  else if (fast_trigger_acquisition_mode == 2)
    fast_trigger_acquisition_mode_string = "DISABLED";
  else
```

```
    fast_trigger_acquisition_mode_string = "UNSET";

  const char *enable_fast_trigger_digitizing_string;
  if (enable_fast_trigger_digitizing == 1)
    enable_fast_trigger_digitizing_string = "YES";
  else if (enable_fast_trigger_digitizing == 0)
    enable_fast_trigger_digitizing_string = "NO";
  else
    enable_fast_trigger_digitizing_string = "UNSET";

  const char *rising_falling_string;
  if (rising_falling == 1)
    rising_falling_string = "FALLING";
  else if (rising_falling == 0)
    rising_falling_string = "RISING";
  else
    rising_falling_string = "UNSET";

  const char *use_interrupt_string;
  if (use_interrupt == 1)
    use_interrupt_string = "YES";
  else if (use_interrupt == 0)
    use_interrupt_string = "NO";
  else
    use_interrupt_string = "UNSET";

  const char *nim_ttl_string;
  if (nim_ttl == 1)
    nim_ttl_string = "TTL";
  else if (nim_ttl == 0)
    nim_ttl_string = "NIM";
  else
    nim_ttl_string = "UNSET";

  const char *enable_input_string;
  if (enable_input == 1)
    enable_input_string = "YES";
  else if (enable_input == 0)
    enable_input_string = "NO";
  else
    enable_input_string = "UNSET";

  const char *self_trigger_acquisition_mode_string;
  if (self_trigger_enable_mask_mode == 0)
    self_trigger_acquisition_mode_string = "ACQUISITION_ONLY";
  else if (self_trigger_enable_mask_mode == 1)
    self_trigger_acquisition_mode_string = "ACQUISITION_AND_TRGOUT";
  else if (self_trigger_enable_mask_mode == 2)
    self_trigger_acquisition_mode_string = "DISABLED";
  else
    self_trigger_acquisition_mode_string = "UNSET";


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT,
        "LinkType\t\t\t\t\t%d\nLinkNumber\t\t\t\t\t%d\nConetNode\t\t\t\t\t%d\nVMEBaseAddress\t\t\t\t\t%d\n",
        LinkType, LinkNumber, ConetNode, VMEBaseAddress);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT,
        "DSR4_Frequency\t\t\t\t\t%d\nOutputFileFormat\t\t\t\t%s\ngnuplot\t\t\t\t\t\t%s\nheader_yes_no\t\t\t\t\t%s\nrecord_length\t\t\t\t\t%d\n",
        DSR4_Frequency, output_file_format_string, gnuplot,
        header_yes_no_string, record_length);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT,
        "DESMod\t\t\t\t\t\t%s\nexternal_trigger_acquisition_mode\t\t%s\nfast_trigger_acquisition_mode\t\t\t%s\n",
        des_mod_string, external_trigger_acquisition_mode_string,
        fast_trigger_acquisition_mode_string);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT,
        "enable_fast_trigger_digitizing\t\t\t%s\nmax_num_events_BLT\t\t\t\t%d\ndecimation_factor\t\t\t\t%d\n",
        enable_fast_trigger_digitizing_string, max_num_events_BLT,
        decimation_factor);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT,
        "post_trigger\t\t\t\t\t%d\nrising_falling\t\t\t\t\t%s\nuse_interrupt\t\t\t\t\t%s\nnim_ttl\t\t\t\t\t
\t%s\n",
        post_trigger, rising_falling_string, use_interrupt_string,
        nim_ttl_string);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT,
        "REGISTER ADDRESS MASK DATA\t\t\t%d %d %d\n", Address_register,
        Data_register, Mask_register);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT,
        "enable_input, dc_offset, trigger_threshold\t%s, %d, %d \n\n",
        enable_input_string, dc_offset, trigger_threshold);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT, "channel enable mask: %x\n",
        channel_enable_mask);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT, "group enable mask: %x\n\n",
        group_enable_mask);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT, "self trigger enable mask: %x\n\n",
        self_trigger_enable_mask);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT,
        "self trigger enable mask mode: %s\n\n",
        self_trigger_acquisition_mode_string);
  output_module->Output(stringa);


  for (i = 0; i < MAXCHANNELOBJECT; i++)
    if (channels[i].set != -1)
      channels[i].PrintChannel ();
  for (i = 0; i < MAXGROUPOBJECT; i++)
    if (groups[i].set != -1)
      groups[i].PrintGroup ();
  for (i = 0; i < MAXFASTOBJECT; i++)
    if (fasts[i].set != -1)
      fasts[i].PrintFast ();

}


void
ConfObject::PrintAll ()
{
  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  char stringa[STANDARDBUFFERLIMIT];

  int i;

  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT,
        "LinkType\t\t\t\t\t%d\nLinkNumber\t\t\t\t\t%d\nConetNode\t\t\t\t\t%d\nVMEBaseAddress\t\t\t\t\t%d\n"
,
        LinkType, LinkNumber, ConetNode,
VMEBaseAddress);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT,
        "DSR4_Frequency\t\t\t\t\t%d\nOutputFileFormat\t\t\t\t%d\ngnuplot\t\t\t\t\t\t%s\nheader_yes_no\t\t\t
\t\t%d\nrecord_length\t\t\t\t\t%d\n",
        DSR4_Frequency, output_file_format,
gnuplot, header_yes_no,
         record_length);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT,
         "DESMod\t\t\t\t\t\t%d\nexternal_trigger_acquisition_mode\t\t%d\nfast_trigger_acquisition_mode\t\t\t
%d\n",
         desmod, external_trigger_acquisition_mode,
         fast_trigger_acquisition_mode);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT,
         "enable_fast_trigger_digitizing\t\t\t%d\nmax_num_events_BLT\t\t\t\t%d\ndecimation_factor\t\t\t\t%d
\n",
         enable_fast_trigger_digitizing,
max_num_events_BLT,
         decimation_factor);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT,
         "post_trigger\t\t\t\t\t%d\nrising_falling\t\t\t\t\t%d\nuse_interrupt\t\t\t\t\t%d\nnim_ttl\t\t\t\t\t
\t%d\n",
         post_trigger, rising_falling, use_interrupt,
nim_ttl);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT,
         "REGISTER ADDRESS MASK DATA\t\t\t%d %d %d\n", Address_register,
         Data_register, Mask_register);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT,
         "enable_input, dc_offset, trigger_threshold\t%d, %d, %d\n\n",
         enable_input, dc_offset, trigger_threshold);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT, "channel enable mask: %x\n",
         channel_enable_mask);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT, "group enable mask: %x\n\n",
         group_enable_mask);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT, "self trigger enable mask: %x\n\n",
         self_trigger_enable_mask);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT,
         "self trigger enable mask mode: %x\n\n",
         self_trigger_enable_mask_mode);
  output_module->Output(stringa);



  for (i = 0; i < MAXCHANNELOBJECT; i++)
    if (channels[i].set != -1)
      channels[i].PrintChannel ();
  for (i = 0; i < MAXGROUPOBJECT; i++)
    if (groups[i].set != -1)
      groups[i].PrintGroup ();
  for (i = 0; i < MAXFASTOBJECT; i++)
    if (fasts[i].set != -1)
      fasts[i].PrintFast ();
}
```

6.23 ConfObject.h File Reference

```
#include "DefineGeneral.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Classes

- class ChannelObject

  The ChannelObject class picks up all the settings taken from the configuration file regarding one channel of the digitizer.
- class GroupObject

  The GroupObject class picks up all the settings taken from the configuration file regarding one group of the digitizer.
- class FastObject

  The FastObject class picks up all the settings taken from the configuration file regarding a TRn input channel of the digitizer (only for x742 series).
- class ConfObject

  The ConfObject class picks up all the settings taken from the configuration file.

6.24 ConfObject.h

```

#include "DefineGeneral.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

class ChannelObject
{
public:

  int set;

  int numChannel;

  int enable_input;

  int dc_offset;

  int trigger_threshold;

  int channel_trigger;

    ChannelObject ();

  void PrintChannel ();
};



class GroupObject
{
public:

  int set;

  int numGroup;

  int enable_input;

  int dc_offset;

  int trigger_threshold;

  int group_trg_enable_mask;

    GroupObject ();

  void PrintGroup ();
};


class FastObject
{
public:

  int set;

  int numFast;

  int dc_offset;

  int trigger_threshold;

    FastObject ();

  void PrintFast ();
};


class ConfObject
{
public:

  int LinkType;

  int LinkNumber;

  int ConetNode;

  int VMEBaseAddress;

  int DSR4_Frequency;

  int output_file_format;

  char *gnuplot;

  int header_yes_no;

  int record_length;

  int test_pattern;

  int desmod;

  int external_trigger_acquisition_mode;

  int fast_trigger_acquisition_mode;

  int enable_fast_trigger_digitizing;

  int max_num_events_BLT;

  int decimation_factor;

  int post_trigger;

  int rising_falling;

  int use_interrupt;

  int nim_ttl;

  int Address_register;

  int Mask_register;

  int Data_register;

  int enable_input;

  int dc_offset;

  int trigger_threshold;

  ChannelObject channels[MAXCHANNELOBJECT];

  GroupObject groups[MAXGROUPOBJECT];

  FastObject fasts[MAXFASTOBJECT];

  int channel_enable_mask;

  int group_enable_mask;
  int self_trigger_enable_mask;

  int self_trigger_enable_mask_mode;

    ConfObject ();

  void PrintAll ();

  void PrintAllHuman ();
};
```

6.25 DefineCommands.h File Reference

This file contains the codes of the commands used to send command via TCP/IP.

Macros

- #define INIT 1
- #define SETUP 2
- #define PRESTART 3
- #define PRESTOP 4
- #define VISTART 5
- #define VISTOP 6
- #define START 7
- #define STOP 8
- #define SEND 9
- #define CLOSE 10
- #define QUIT 11
- #define RAWSTART 12
- #define RAWSTOP 13
- #define PRINT 14
- #define CHECK 16
- #define CHKCONF 17
- #define MORE 18
- #define CHANGECONF 21
- #define CHANGEDATA 22
- #define CHANGELOG 23
- #define WRITEREGISTER 24
- #define READREGISTER 25
- #define PRINTFILES 28
- #define STATUS 30

6.25.1 Detailed Description

This file contains the codes of the commands used to send command via TCP/IP.

Author

Daniele Berto

Definition in file DefineCommands.h.

6.25.2 Macro Definition Documentation

6.25.2.1 #define CHANGECONF 21

Definition at line 24 of file DefineCommands.h.

6.25.2.2 #define CHANGEDATA 22

Definition at line 25 of file DefineCommands.h.

6.25.2.3 #define CHANGELOG 23

Definition at line 26 of file DefineCommands.h.

6.25.2.4 #define CHECK 16

Definition at line 21 of file DefineCommands.h.

6.25.2.5 #define CHKCONF 17

Definition at line 22 of file DefineCommands.h.

6.25.2.6 #define CLOSE 10

Definition at line 16 of file DefineCommands.h.

6.25.2.7 #define INIT 1

Definition at line 7 of file DefineCommands.h.

6.25.2.8 #define MORE 18

Definition at line 23 of file DefineCommands.h.

6.25.2.9 #define PRESTART 3

Definition at line 9 of file DefineCommands.h.

6.25.2.10 #define PRESTOP 4

Definition at line 10 of file DefineCommands.h.

6.25.2.11 #define PRINT 14

Definition at line 20 of file DefineCommands.h.

6.25.2.12 #define PRINTFILES 28

Definition at line 29 of file DefineCommands.h.

6.25.2.13 #define QUIT 11

Definition at line 17 of file DefineCommands.h.

6.25.2.14 #define RAWSTART 12

Definition at line 18 of file DefineCommands.h.

6.25.2.15 #define RAWSTOP 13

Definition at line 19 of file DefineCommands.h.

6.25.2.16 #define READREGISTER 25

Definition at line 28 of file DefineCommands.h.

6.25.2.17 #define SEND 9

Definition at line 15 of file DefineCommands.h.

6.25.2.18 #define SETUP 2

Definition at line 8 of file DefineCommands.h.

6.25.2.19 #define START 7

Definition at line 13 of file DefineCommands.h.

6.25.2.20 #define STATUS 30

Definition at line 30 of file DefineCommands.h.

6.25.2.21 #define STOP 8

Definition at line 14 of file DefineCommands.h.

6.25.2.22 #define VISTART 5

Definition at line 11 of file DefineCommands.h.

6.25.2.23 #define VISTOP 6

Definition at line 12 of file DefineCommands.h.

6.25.2.24 #define WRITEREGISTER 24

Definition at line 27 of file DefineCommands.h.

6.26 DefineCommands.h

```
00001
00007 #define INIT 1
00008 #define SETUP 2
00009 #define PRESTART 3
```

```
#define PRESTOP 4
#define VISTART 5
#define VISTOP 6
#define START 7
#define STOP 8
#define SEND 9
#define CLOSE 10
#define QUIT 11
#define RAWSTART 12
#define RAWSTOP 13
#define PRINT 14
#define CHECK 16
#define CHKCONF 17
#define MORE 18
#define CHANGECONF 21
#define CHANGEDATA 22
#define CHANGELOG 23
#define WRITEREGISTER 24
#define READREGISTER 25
#define PRINTFILES 28
#define STATUS 30
```

6.27 DefineGeneral.h File Reference

This file contains same parameters used by the program.

Macros

- #define STANDARDBUFFERLIMIT 1000
- #define RAWDATAQUEUE 10

  The dimension of the rawdata thread queue.
- #define VISUALIZATIONQUEUE 10

  The dimension of the visualization thread queue.
- #define PREPROCESSINGQUEUE 10

  The dimension of the preprocessing thread queue.
- #define MAXCHANNELOBJECT 64

  The max number of channels managed by the program.
- #define MAXGROUPOBJECT 8

  The max number of groups managed by the program.
- #define MAXFASTOBJECT 2

  The max number of fast channels managed by the program.
- #define MAXCOMMAND 3

  The dimension of the command buffer queue (see CommunicationObject class).
- #define ALL 2
- #define ONLYUSER 0
- #define ONLYTCP 1

6.27.1 Detailed Description

This file contains same parameters used by the program. In particular, it contains the dimension of the queues used by the aquisition threads.

Author

Daniele Berto

Definition in file DefineGeneral.h.

6.27.2 Macro Definition Documentation

6.27.2.1 #define ALL 2

Definition at line 31 of file DefineGeneral.h.

6.27.2.2 #define MAXCHANNELOBJECT 64

The max number of channels managed by the program.

Definition at line 19 of file DefineGeneral.h.

6.27.2.3 #define MAXCOMMAND 3

The dimension of the command buffer queue (see CommunicationObject class).

Definition at line 28 of file DefineGeneral.h.

6.27.2.4 #define MAXFASTOBJECT 2

The max number of fast channels managed by the program.

Definition at line 25 of file DefineGeneral.h.

6.27.2.5 #define MAXGROUPOBJECT 8

The max number of groups managed by the program.

Definition at line 22 of file DefineGeneral.h.

6.27.2.6 #define ONLYTCP 1

Definition at line 33 of file DefineGeneral.h.

6.27.2.7 #define ONLYUSER 0

Definition at line 32 of file DefineGeneral.h.

6.27.2.8 #define PREPROCESSINGQUEUE 10

The dimension of the preprocessing thread queue.

Definition at line 16 of file DefineGeneral.h.

6.27.2.9 #define RAWDATAQUEUE 10

The dimension of the rawdata thread queue.

Definition at line 10 of file DefineGeneral.h.

6.27.2.10 #define STANDARDBUFFERLIMIT 1000

Definition at line 7 of file DefineGeneral.h.

6.27.2.11 #define VISUALIZATIONQUEUE 10

The dimension of the visualization thread queue.

Definition at line 13 of file DefineGeneral.h.

6.28 DefineGeneral.h

```
00001
00007 #define STANDARDBUFFERLIMIT 1000
00008
00010 #define RAWDATAQUEUE 10
00011
00013 #define VISUALIZATIONQUEUE 10
00014
00016 #define PREPROCESSINGQUEUE 10
00017
00019 #define MAXCHANNELOBJECT 64
00020
00022 #define MAXGROUPOBJECT 8
00023
00025 #define MAXFASTOBJECT 2
00026
00028 #define MAXCOMMAND 3
00029
00030 //The output modality chosen by the user with the "-m" flag when he launches the program.
00031 #define ALL 2
00032 #define ONLYUSER 0
00033 #define ONLYTCP 1
```

6.29 DigitizerErrorObject.cpp File Reference

```
#include "DigitizerErrorObject.h"
#include "DefineGeneral.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <CAENDigitizer.h>
#include <sys/time.h>
#include <time.h>
```

6.30 DigitizerErrorObject.cpp

```
00001
00005 #include "DigitizerErrorObject.h"
00006 #include "DefineGeneral.h"
00007 #include <stdio.h>
00008 #include <stdlib.h>
00009 #include <string.h>
00010 #include <CAENDigitizer.h>
00011 #include<sys/time.h>
00012 #include<time.h>
00013
00014 void
00015 DigitizerErrorObject::DigitizerErrorObjectDebugging (
      CAEN_DGTZ_ErrorCode
00016                              ritorno,
00017                              const char *file,
00018                              const char *func,
00019                              int line)
00020 {
00021   switch (ritorno)
00022     {
00023
00024     case (CAEN_DGTZ_Success):
00025       fprintf (stderr,
00026            "File: %s Function: %s Line: %d ErrorCode: %d Operation completed successful\n",
00027            file, func, line, ritorno);
```

```
      break;

    case (CAEN_DGTZ_CommError):
      fprintf (stderr,
           "File: %s Function: %s Line: %d ErrorCode: %d Communication error\n",
           file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_GenericError):
      fprintf (stderr,
           "File: %s Function: %s Line: %d ErrorCode: %d Unspecified error\n",
           file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_InvalidParam):
      fprintf (stderr,
           "File: %s Function: %s Line: %d ErrorCode: %d Invalid parameter\n",
           file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_InvalidLinkType):
      fprintf (stderr,
           "File: %s Function: %s Line: %d ErrorCode: %d Invalid Link Type\n",
           file, func, line, ritorno);
      break;

    case (-5):
      fprintf (stderr,
           "File: %s Function: %s Line: %d ErrorCode:%d Invalid device handler\n",
           file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_MaxDevicesError):
      fprintf (stderr,
           "File: %s Function: %s Line: %d ErrorCode: %d Maximum number of devices exceeded\n",
           file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_BadBoardType):
      fprintf (stderr,
           "File: %s Function: %s Line: %d ErrorCode: %d Operation non allowed on this type of board\n",
           file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_BadInterruptLev):
      fprintf (stderr,
           "File: %s Function: %s Line: %d ErrorCode: %d The interrupt level is not allowed\n",
           file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_BadEventNumber):
      fprintf (stderr,
           "File: %s Function: %s Line: %d ErrorCode: %d The event number is bad\n",
           file, func, line, ritorno);
      break;

    case (-10):
      fprintf (stderr,
           "File: %s Function: %s Line: %d ErrorCode:%d Unable to read the regitry\n",
           file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_ChannelBusy):
      fprintf (stderr,
           "File: %s Function: %s Line: %d ErrorCode: %d The channel number is invalid\n",
           file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_FPIOModeInvalid):
      fprintf (stderr,
           "File: %s Function: %s Line: %d ErrorCode: %d Invalid FPIO mode\n",
           file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_WrongAcqMode):
      fprintf (stderr,
           "File: %s Function: %s Line: %d ErrorCode: %d Wrong acquisition mode\n",
           file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_FunctionNotAllowed):
      fprintf (stderr,
           "File: %s Function: %s Line: %d ErrorCode: %d This function is not allowed for this module\n",
           file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_Timeout):
      fprintf (stderr,
            "File: %s Function: %s Line: %d ErrorCode: %d Communication Timeout\n",
            file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_InvalidBuffer):
      fprintf (stderr,
            "File: %s Function: %s Line: %d ErrorCode: %d The buffer is invalid\n",
            file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_EventNotFound):
      fprintf (stderr,
            "File: %s Function: %s Line: %d ErrorCode: %d The event is not found\n",
            file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_InvalidEvent):
      fprintf (stderr,
            "File: %s Function: %s Line: %d ErrorCode: %d The event is invalid\n",
            file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_OutOfMemory):
      fprintf (stderr,
            "File: %s Function: %s Line: %d ErrorCode: %d Out of memory\n",
            file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_CalibrationError):
      fprintf (stderr,
            "File: %s Function: %s Line: %d ErrorCode: %d Unable to calibrate the board\n",
            file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_DigitizerNotFound):
      fprintf (stderr,
            "File: %s Function: %s Line: %d ErrorCode: %d Unable to open the digitizer\n",
            file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_DigitizerAlreadyOpen):
      fprintf (stderr,
            "File: %s Function: %s Line: %d ErrorCode: %d The Digitizer is already open\n",
            file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_DigitizerNotReady):
      fprintf (stderr,
            "File: %s Function: %s Line: %d ErrorCode: %d The Digitizer is not ready to operate\n",
            file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_InterruptNotConfigured):
      fprintf (stderr,
            "File: %s Function: %s Line: %d ErrorCode: %d The Digitizer has not the IRQ configured\n",
            file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_DigitizerMemoryCorrupted):
      fprintf (stderr,
            "File: %s Function: %s Line: %d ErrorCode: %d The Digitizer flash memory is corrupted\n",
            file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_DPPFirmwareNotSupported):
      fprintf (stderr,
            "File: %s Function: %s Line: %d ErrorCode: %d The digitizer DPP firmware is not supported in
 this lib version\n",
            file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_InvalidLicense):
      fprintf (stderr,
            "File: %s Function: %s Line: %d ErrorCode: %d Invalid Firmware License\n",
            file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_InvalidDigitizerStatus):
      fprintf (stderr,
            "File: %s Function: %s Line: %d ErrorCode: %d The digitizer is found in a corrupted status\n",
            file, func, line, ritorno);
      break;

    case (CAEN_DGTZ_UnsupportedTrace):
      fprintf (stderr,
            "File: %s Function: %s Line: %d ErrorCode: %d The given trace is not supported by the digitizer
```

```
\n",
                file, func, line, ritorno);
          break;

        case (CAEN_DGTZ_InvalidProbe):
          fprintf (stderr,
                "File: %s Function: %s Line: %d ErrorCode: %d The given probe is not supported for the given
       digitizer's trace\n",
                file, func, line, ritorno);
          break;

        case (CAEN_DGTZ_NotYetImplemented):
          fprintf (stderr,
                "File: %s Function: %s Line: %d ErrorCode: %d The function is not yet implemented\n",
                file, func, line, ritorno);
          break;
        }

}

void
DigitizerErrorObject::DigitizerErrorObjectPrintError
  (CAEN_DGTZ_ErrorCode ritorno)
{
  switch (ritorno)
    {

    case (CAEN_DGTZ_Success):
      fprintf (stderr, "ErrorCode: %d Operation completed successful\n",
           ritorno);
      break;

    case (CAEN_DGTZ_CommError):
      fprintf (stderr, "ErrorCode: %d Communication error\n", ritorno);
      break;

    case (CAEN_DGTZ_GenericError):
      fprintf (stderr, "ErrorCode: %d Unspecified error\n", ritorno);
      break;

    case (CAEN_DGTZ_InvalidParam):
      fprintf (stderr, "ErrorCode: %d Invalid parameter\n", ritorno);
      break;

    case (CAEN_DGTZ_InvalidLinkType):
      fprintf (stderr, "ErrorCode: %d Invalid Link Type\n", ritorno);
      break;

    case (-5):
      fprintf (stderr, "ErrorCode:%d Invalid device handler\n", ritorno);
      break;

    case (CAEN_DGTZ_MaxDevicesError):
      fprintf (stderr, "ErrorCode: %d Maximum number of devices exceeded\n",
           ritorno);
      break;

    case (CAEN_DGTZ_BadBoardType):
      fprintf (stderr,
           "ErrorCode: %d Operation non allowed on this type of board\n",
           ritorno);
      break;

    case (CAEN_DGTZ_BadInterruptLev):
      fprintf (stderr, "ErrorCode: %d The interrupt level is not allowed\n",
           ritorno);
      break;

    case (CAEN_DGTZ_BadEventNumber):
      fprintf (stderr, "ErrorCode: %d The event number is bad\n", ritorno);
      break;

    case (-10):
      fprintf (stderr, "ErrorCode:%d Unable to read the regitry\n", ritorno);
      break;

    case (CAEN_DGTZ_ChannelBusy):
      fprintf (stderr, "ErrorCode: %d The channel number is invalid\n",
           ritorno);
      break;

    case (CAEN_DGTZ_FPIOModeInvalid):
      fprintf (stderr, "ErrorCode: %d Invalid FPIO mode\n", ritorno);
      break;

    case (CAEN_DGTZ_WrongAcqMode):
      fprintf (stderr, "ErrorCode: %d Wrong acquisition mode\n", ritorno);
      break;

    case (CAEN_DGTZ_FunctionNotAllowed):
      fprintf (stderr,
           "ErrorCode: %d This function is not allowed for this module\n",
           ritorno);
      break;

    case (CAEN_DGTZ_Timeout):
      fprintf (stderr, "ErrorCode: %d Communication Timeout\n", ritorno);
      break;

    case (CAEN_DGTZ_InvalidBuffer):
      fprintf (stderr, "ErrorCode: %d The buffer is invalid\n", ritorno);
      break;

    case (CAEN_DGTZ_EventNotFound):
      fprintf (stderr, "ErrorCode: %d The event is not found\n", ritorno);
      break;

    case (CAEN_DGTZ_InvalidEvent):
      fprintf (stderr, "ErrorCode: %d The event is invalid\n", ritorno);
      break;

    case (CAEN_DGTZ_OutOfMemory):
      fprintf (stderr, "ErrorCode: %d Out of memory\n", ritorno);
      break;

    case (CAEN_DGTZ_CalibrationError):
      fprintf (stderr, "ErrorCode: %d Unable to calibrate the board\n",
           ritorno);
      break;

    case (CAEN_DGTZ_DigitizerNotFound):
      fprintf (stderr, "ErrorCode: %d Unable to open the digitizer\n",
           ritorno);
      break;

    case (CAEN_DGTZ_DigitizerAlreadyOpen):
      fprintf (stderr, "ErrorCode: %d The Digitizer is already open\n",
           ritorno);
      break;

    case (CAEN_DGTZ_DigitizerNotReady):
      fprintf (stderr,
           "ErrorCode: %d The Digitizer is not ready to operate\n",
           ritorno);
      break;

    case (CAEN_DGTZ_InterruptNotConfigured):
      fprintf (stderr,
           "ErrorCode: %d The Digitizer has not the IRQ configured\n",
           ritorno);
      break;

    case (CAEN_DGTZ_DigitizerMemoryCorrupted):
      fprintf (stderr,
           "ErrorCode: %d The Digitizer flash memory is corrupted\n",
           ritorno);
      break;

    case (CAEN_DGTZ_DPPFirmwareNotSupported):
      fprintf (stderr,
           "ErrorCode: %d The digitizer DPP firmware is not supported in this lib version\n",
           ritorno);
      break;

    case (CAEN_DGTZ_InvalidLicense):
      fprintf (stderr, "ErrorCode: %d Invalid Firmware License\n", ritorno);
      break;

    case (CAEN_DGTZ_InvalidDigitizerStatus):
      fprintf (stderr,
           "ErrorCode: %d The digitizer is found in a corrupted status\n",
           ritorno);
      break;

    case (CAEN_DGTZ_UnsupportedTrace):
      fprintf (stderr,
           "ErrorCode: %d The given trace is not supported by the digitizer\n",
           ritorno);
      break;

    case (CAEN_DGTZ_InvalidProbe):
      fprintf (stderr,
           "ErrorCode: %d The given probe is not supported for the given digitizer's trace\n",
           ritorno);
      break;

    case (CAEN_DGTZ_NotYetImplemented):
      fprintf (stderr, "ErrorCode: %d The function is not yet implemented\n",
           ritorno);
      break;
    }
}


int
DigitizerErrorObject::DigitizerErrorObjectDebuggingLog
 (CAEN_DGTZ_ErrorCode
                              ritorno,
                              const char *file,
                              const char *func,
                              int line,
                              FILE * log_file)
{

  char ts[100];
  time_t t;
  t = time (NULL);
  ctime_r (&t, ts);
  ts[strlen (ts) - 1] = '\0';

  switch (ritorno)
    {

    case (CAEN_DGTZ_Success):
      fprintf (log_file,
           "%s File: %s Function: %s Line: %d ErrorCode: %d Operation completed successfully\n",
           ts, file, func, line, ritorno);
      fflush (log_file);
      return 0;
      break;

    case (CAEN_DGTZ_CommError):
      fprintf (log_file,
           "%s File: %s Function: %s Line: %d ErrorCode: %d Communication error\n",
           ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_GenericError):
      fprintf (log_file,
           "%s File: %s Function: %s Line: %d ErrorCode: %d Unspecified error\n",
           ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_InvalidParam):
      fprintf (log_file,
           "%s File: %s Function: %s Line: %d ErrorCode: %d Invalid parameter\n",
           ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_InvalidLinkType):
      fprintf (log_file,
           "%s File: %s Function: %s Line: %d ErrorCode: %d Invalid Link Type\n",
           ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (-5):
      fprintf (log_file,
           "%s File: %s Function: %s Line: %d ErrorCode:%d Invalid device handler\n",
           ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_MaxDevicesError):
      fprintf (log_file,
           "%s File: %s Function: %s Line: %d ErrorCode: %d Maximum number of devices exceeded\n",
           ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_BadBoardType):
      fprintf (log_file,
            "%s File: %s Function: %s Line: %d ErrorCode: %d Operation non allowed on this type of board\n",
            ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_BadInterruptLev):
      fprintf (log_file,
            "%s File: %s Function: %s Line: %d ErrorCode: %d The interrupt level is not allowed\n",
            ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_BadEventNumber):
      fprintf (log_file,
            "%s File: %s Function: %s Line: %d ErrorCode: %d The event number is bad\n",
            ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (-10):
      fprintf (log_file,
            "%s File: %s Function: %s Line: %d ErrorCode:%d Unable to read the regitry\n",
            ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_ChannelBusy):
      fprintf (log_file,
            "%s File: %s Function: %s Line: %d ErrorCode: %d The channel number is invalid\n",
            ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_FPIOModeInvalid):
      fprintf (log_file,
            "%s File: %s Function: %s Line: %d ErrorCode: %d Invalid FPIO mode\n",
            ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_WrongAcqMode):
      fprintf (log_file,
            "%s File: %s Function: %s Line: %d ErrorCode: %d Wrong acquisition mode\n",
            ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_FunctionNotAllowed):
      fprintf (log_file,
            "%s File: %s Function: %s Line: %d ErrorCode: %d This function is not allowed for this module\n"
,
            ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_Timeout):
      fprintf (log_file,
            "%s File: %s Function: %s Line: %d ErrorCode: %d Communication Timeout\n",
            ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_InvalidBuffer):
      fprintf (log_file,
            "%s File: %s Function: %s Line: %d ErrorCode: %d The buffer is invalid\n",
            ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_EventNotFound):
      fprintf (log_file,
            "%s File: %s Function: %s Line: %d ErrorCode: %d The event is not found\n",
            ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_InvalidEvent):
      fprintf (log_file,
           "%s File: %s Function: %s Line: %d ErrorCode: %d The event is invalid\n",
           ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_OutOfMemory):
      fprintf (log_file,
           "%s File: %s Function: %s Line: %d ErrorCode: %d Out of memory\n",
           ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_CalibrationError):
      fprintf (log_file,
           "%s File: %s Function: %s Line: %d ErrorCode: %d Unable to calibrate the board\n",
           ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_DigitizerNotFound):
      fprintf (log_file,
           "%s File: %s Function: %s Line: %d ErrorCode: %d Unable to open the digitizer\n",
           ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_DigitizerAlreadyOpen):
      fprintf (log_file,
           "%s File: %s Function: %s Line: %d ErrorCode: %d The Digitizer is already open\n",
           ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_DigitizerNotReady):
      fprintf (log_file,
           "%s File: %s Function: %s Line: %d ErrorCode: %d The Digitizer is not ready to operate\n",
           ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_InterruptNotConfigured):
      fprintf (log_file,
           "%s File: %s Function: %s Line: %d ErrorCode: %d The Digitizer has not the IRQ configured\n",
           ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_DigitizerMemoryCorrupted):
      fprintf (log_file,
           "%s File: %s Function: %s Line: %d ErrorCode: %d The Digitizer flash memory is corrupted\n",
           ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_DPPFirmwareNotSupported):
      fprintf (log_file,
           "%s File: %s Function: %s Line: %d ErrorCode: %d The digitizer DPP firmware is not supported in
this lib version\n",
           ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_InvalidLicense):
      fprintf (log_file,
           "%s File: %s Function: %s Line: %d ErrorCode: %d Invalid Firmware License\n",
           ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_InvalidDigitizerStatus):
      fprintf (log_file,
           "%s File: %s Function: %s Line: %d ErrorCode: %d The digitizer is found in a corrupted status\n"
,
           ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_UnsupportedTrace):
      fprintf (log_file,
           "%s File: %s Function: %s Line: %d ErrorCode: %d The given trace is not supported by the
 digitizer\n",
           ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_InvalidProbe):
      fprintf (log_file,
           "%s File: %s Function: %s Line: %d ErrorCode: %d The given probe is not supported for the given
 digitizer's trace\n",
           ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;

    case (CAEN_DGTZ_NotYetImplemented):
      fprintf (log_file,
           "%s File: %s Function: %s Line: %d ErrorCode: %d The function is not yet implemented\n",
           ts, file, func, line, ritorno);
      fflush (log_file);
      return 1;
      break;
    }

}
```

6.31 DigitizerErrorObject.h File Reference

```
#include <stdio.h>
#include <CAENDigitizer.h>
```

Classes

- class DigitizerErrorObject

  The DigitizerErrorObject class contains two methods that print the meaning of the CAEN_DGTZ_ErrorCode.

6.32 DigitizerErrorObject.h

```

#include <stdio.h>
#include <CAENDigitizer.h>

class DigitizerErrorObject
{
public:

  void DigitizerErrorObjectDebugging (CAEN_DGTZ_ErrorCode ritorno,
                     const char *file, const char *func,
                     int line);
  void DigitizerErrorObjectPrintError (CAEN_DGTZ_ErrorCode ritorno);


  int DigitizerErrorObjectDebuggingLog (CAEN_DGTZ_ErrorCode ritorno,
                    const char *file, const char *func,
                    int line, FILE * log_file);
};
```

6.33 DigitizerFlowControl.cpp File Reference

```
#include "DefineGeneral.h"
```

```
#include "DefineCommands.h"
#include "TcpUser.h"
#include "ConfObject.h"
#include "DigitizerErrorObject.h"
#include "LogFile.h"
#include "DigitizerObject.h"
#include "DigitizerObjectGeneric.h"
#include "RawData.h"
#include "DigitizerStateMachine.h"
#include "ApplicationSetup.h"
#include "Input.h"
#include "CommunicationObject.h"
#include "DigitizerFlowControl.h"
#include "ConfigurationConsistence.h"
#include "AnalizzatoreUtils.h"
#include "Analizzatore.h"
#include "OutputModule.h"
#include <CAENDigitizer.h>
#include <stdio.h>
#include <regex.h>
#include <stdbool.h>
```

6.34 DigitizerFlowControl.cpp

```

#include "DefineGeneral.h"
#include "DefineCommands.h"
#include "TcpUser.h"
#include "ConfObject.h"
#include "DigitizerErrorObject.h"
#include "LogFile.h"
#include "DigitizerObject.h"
#include "DigitizerObjectGeneric.h"
#include "RawData.h"
#include "DigitizerStateMachine.h"
#include "ApplicationSetup.h"
#include "Input.h"
#include "CommunicationObject.h"
#include "DigitizerFlowControl.h"
#include "ConfigurationConsistence.h"
#include "AnalizzatoreUtils.h"
#include "Analizzatore.h"
#include "OutputModule.h"
#include <CAENDigitizer.h>
#include <stdio.h>
#include <regex.h>
#include <stdbool.h>

pthread_cond_t DigitizerFlowControl::input_flow_cond;

pthread_mutex_t DigitizerFlowControl::input_flow_mutex;

// L'oggetto di tipo DigitizerFlowControl viene inizializzato ottenendo un riferimento all'oggetto
 ApplicationSetup.
DigitizerFlowControl::DigitizerFlowControl ()
{
  pthread_mutex_init (&input_flow_mutex, NULL);
  pthread_cond_init (&input_flow_cond, NULL);
  digitizer_flow_control_application_setup =
ApplicationSetup::Instance ();
}


void
DigitizerFlowControl::DigitizerFlowControlStart ()
{

//Viene ottenuto il riferimento ad OutputModule.
  OutputModule *output_module;
  output_module = OutputModule::Instance ();

//E' ottenuto un riferimento al singleton LogFile. LogFile permette di stampare (in modo verboso) il
```

```
risultato delle funzioni della libreria CAENDigitizer nel file di log
//indicato dal flag -l [logfilepath]. Se l'utente non ne indica uno, viene usato quello di default. Il contenuto del log file puo' essere visualizzato digitando il comando "more".
//Questo permette di controllare rapidamente se gli altri comandi che interagiscono col digitizer hanno avuto successo.
  LogFile *logfile;
  logfile = LogFile::Instance ();

//Variabili di supporto
  char *my_punt;
  char *conf_file_string;
  char *data_file_string;
  char *log_file_string;
  int i;
  int Address = 0;
  int Data = 0;
  char support_string[STANDARDBUFFERLIMIT];
  bzero (support_string, STANDARDBUFFERLIMIT);
  int support_flag1 = 0;

//Impostazione del log file. Il metodo LogFileSet apre il log file indicato dal flag -l [logfilepath] o quello di default.
  logfile->LogFileSet (digitizer_flow_control_application_setup->
              application_setup_log_file_path);

//L'oggetto di tipo ConfigurationConsistence serve per controllare se nel file di configurazione sono presenti errori gravi (come l'omissione dell'attributo OPEN).
//Questo tipo di controllo e' possibile farlo con il comando "check".
  ConfigurationConsistence my_consistence;

//Se il metodo ConfigurationConsistenceConfFileSetupEssentialWithInitCheckNoPrint torna un valore diverso da zero vuol dire che ha trovato un errore nel file di configurazione.
  if (my_consistence.ConfigurationConsistenceConfFileSetupEssentialWithInitCheckNoPrint (digitizer_flow_control_application_setup->application_setup_conf_file_path))
    {
      fprintf (stderr, "%s: Some error detected in %s config file\n",
           __func__,
           digitizer_flow_control_application_setup->application_setup_conf_file_path);
    }

//DigitizerStateMachine e' il tipo di oggetto che comunica col digitizer. E', insieme a DigitizerFlowControl, il tipo di oggetto piu' delicato perche' lancia e gestisce
//il thread che comunica col digitizer e quelli che trattano i dati letti.
  DigitizerStateMachine miodigitizer;

//buffer contiene il comando fetchato da stdin o da tcp.
  char *buffer;
  buffer = (char *) malloc (STANDARDBUFFERLIMIT);
  TcpUser command_received_by_user;

//A seconda della modalita' di interazione scelta dall'utente (-m [all|user|tcp]) occorre inizializzare gli oggetti per fetchare l'input.

//Se l'utente ha lanciato il programma con il flag -m all, vuol dire che l'input puo' essere inviato sia da stdin sia da tcp.
  if (digitizer_flow_control_application_setup->input_mode == ALL)
    {

//Inizializzo CommunicationObject che raccoglie l'input via TCP.
      mioTCP.CommunicationObjectInit ();

    }   //if (digitizer_flow_control_application_setup->input_mode == ALL)

//Se l'utente ha lanciato il programma con il flag -m user, vuol dire che l'input puo' essere inviato solo da stdin.
  else if (digitizer_flow_control_application_setup->input_mode == ONLYUSER)
    {
    ;
    }   //if (digitizer_flow_control_application_setup->input_mode == ONLYUSER)

//Se l'utente ha lanciato il programma con il flag -m tcp, vuol dire che l'input puo' essere inviato solo via TCP.
  else if (digitizer_flow_control_application_setup->input_mode == ONLYTCP)
    {
//Inizializzo CommunicationObject che raccoglie l'input via TCP.
      mioTCP.CommunicationObjectInit ();
    }   //if (digitizer_flow_control_application_setup->input_mode == ONLYTCP)

//Se il programma e' stato eseguito in una modalita' diversa da ONLYTCP, stampo la stringa di benvenuto.
//I controlli vengono fatti dai singleton StdOut e TcpUserArray.
```

```
  output_module->Output("objectDump, press help to get the available command list\n");

//Inizia il ciclo di fetch-decode-execute-writeback del programma.
  while (1)
    {

//Se il programma e' stato lanciato in una modalita' diversa da quella ONLYTCP e' effettuato un flush del
 stdout per stampare l'eventuale output bufferizzato.
//Per evitare la bufferizzazione e' anche possibile stampare su stderr invece che su stdout, ma
 risulterebbe essere un utilizzo non appropriato di stderr.
      if (digitizer_flow_control_application_setup->
input_mode != ONLYTCP)
    {
      fflush (stdout);
    }

//Settiamo a zero buffer: in esso viene salvato il contenuto dell'input ottenuto dagli oggetti di tipo
 CommunicationObject (per il TCP) e Input (per l'stdout).
      bzero (buffer, STANDARDBUFFERLIMIT);

//Azioni da intraprende per fetchare l'input se il programma e' stato lanciato sia in modalita' tcp che in
 modalita' user
      if (digitizer_flow_control_application_setup->
input_mode == ALL)
    {

//Chiedo l'input da stdin. Se non c'e' il programma prosegue e in buffer e' salvata la stringa vuota.
      mioinput.GetInput (buffer);

//Se e' stato ricevuto un input dall'stdin, e' stampata una stringa che lo riproduce.
      if (strlen (buffer) != 0)
        {

          output_module->OutputModuleStdoutOn();
          output_module->OutputModuleSockidOff();

        }

//Chiedo l'input inviato da TCP.
      command_received_by_user = mioTCP.GetCommand ();

//Se e' stato ricevuto un input via tcp.
      if (command_received_by_user.command_sent_by_user != 0)
        {

//E' stampata una stringa che scrive l'input ricevuto e il sockid di chi l'ha inviato.
//Questo avviene solo se il programma e' lanciato in modalita' -m all: se e' lanciato in modalita' -m tcp,
 esso e' eseguito in modalita' demone.
//Di conseguenza, in quest'ultimo caso non stampa niente a video.
          fprintf (stdout, "Command sent by user %d: %d which means %s\n",
               command_received_by_user.user_sockid,
               command_received_by_user.command_sent_by_user,
               DigitizerFlowControl::
               ParseCommand (command_received_by_user.
                     command_sent_by_user));

          output_module->OutputModuleStdoutOff();
          output_module->OutputModuleSockidOn(command_received_by_user.
user_sockid);

        }
    }  //if (digitizer_flow_control_application_setup->input_mode == ALL)


//Azioni da intraprende per fetchare l'input se il programma e' stato lanciato in modalita' user.
      else if (digitizer_flow_control_application_setup->
input_mode ==
           ONLYUSER)
    {

//Chiedo l'input dall'stdin. Se non c'e' il programma prosegue e in buffer e' salvata la stringa vuota.
      mioinput.GetInput (buffer);

//Se e' stato ricevuto un input dall'stdin, e' stampata una stringa che lo riproduce.
      if (strlen (buffer) != 0)
        {
          output_module->OutputModuleStdoutOn();
          output_module->OutputModuleSockidOff();

        }
    }  //if (digitizer_flow_control_application_setup->input_mode == ONLYUSER)


//Azioni da intraprende per fetchare l'input se il programma e' stato lanciato in modalita' tcp.
      else if (digitizer_flow_control_application_setup->
input_mode ==
           ONLYTCP)
    {

//Chiedo l'input inviato da TCP.
      command_received_by_user = mioTCP.GetCommand ();

//Se e' stato ricevuto un input via TCP.
      if (command_received_by_user.command_sent_by_user != 0)
        {

          output_module->OutputModuleStdoutOff();
          output_module->OutputModuleSockidOn(command_received_by_user.
user_sockid);

        }

    }  //if (digitizer_flow_control_application_setup->input_mode == ONLYTCP)

      pthread_mutex_lock (&input_flow_mutex);

      //Se e' stato ricevuto un input via tcp o via stdin.
      if (strlen (buffer) != 0
      || command_received_by_user.command_sent_by_user != 0)
    {
      //Se e' stato ricevuto il comando di exit o di quit
      if (DigitizerFlowControl::reg_matches (buffer, "^[Ee][Xx][iI][tT][
\t]*$") || DigitizerFlowControl::reg_matches (buffer, "^[qQ][uU][iI][tT]$")
|| command_received_by_user.command_sent_by_user == QUIT)    //11==quit
        {
          output_module->OutputModuleStdoutOn();

          //Spengo CommunicationObject
          if (digitizer_flow_control_application_setup->
input_mode !=
          ONLYUSER)
        mioTCP.Finish ();

          //Spengo Input.
          if (digitizer_flow_control_application_setup->
input_mode !=
          ONLYTCP)
        mioinput.Finish ();

          //Spengo DigitizerStateMachine.
          miodigitizer.DigitizerStateMachineQuit ();
          return;
        }

      //Se e' stato ricevuto il comando init.
      else if (DigitizerFlowControl::
           reg_matches (buffer, "^[iI][Nn][Ii][Tt][ \t]*$")
           || command_received_by_user.command_sent_by_user ==
INIT)
        {
          //Controllo se il file di configurazione contiene l'attributo OPEN e se e' ben settato. Il
 NoPrint del nome del metodo indica che non viene stampato a video
          //il parametro OPEN del file di configurazione.
          if (!my_consistence.
          ConfigurationConsistenceConfFileInitNoPrint
          (digitizer_flow_control_application_setup->
           application_setup_conf_file_path))
        {
          //Se non ci sono errori nell'attributo OPEN del file di configurazione, il digitizer viene
 aperto.
          output_module->Output("Digitizer initialization...\n");

          logfile->LogFileMessageOn();

          miodigitizer.
            DigitizerStateMachineInit
            (digitizer_flow_control_application_setup->
             application_setup_conf_file_path);

          logfile->LogFileMessageOff();
        }
          else
        {
          //Altrimenti non viene intrapresa nessuna azione
          output_module->Output("Some error occured, I cannot execute init command\n");
        }
        }

      //Se e' stato ricevuto il comando setup.
      else if (DigitizerFlowControl::
           reg_matches (buffer, "^[Ss][Ee][Tt][Uu][Pp][ \t]*$")
           || command_received_by_user.command_sent_by_user ==
SETUP)
        {
```

```
        //Controllo se il file di configurazione contiene i parametri fondamentali per settare il digitizer.
        if (!my_consistence.
        ConfigurationConsistenceConfFileSetupEssentialWithInitCheckNoPrint
        (digitizer_flow_control_application_setup->
         application_setup_conf_file_path))
      {
        //Se non ci sono errori nell'attributo OPEN del file di configurazione, il digitizer viene impostato.
        output_module->Output("Digitizer setupping...\n");

        logfile->LogFileMessageOn();

        miodigitizer.
          DigitizerStateMachineSetup
          (digitizer_flow_control_application_setup->
           application_setup_conf_file_path);

        logfile->LogFileMessageOff();
      }
      }

    //Se e' stato ricevuto il comando start.
    else if (DigitizerFlowControl::
         reg_matches (buffer, "^[sS][tT][aA][rR][tT][ \t]*$")
         || command_received_by_user.command_sent_by_user == START)
      {
        //Il programma inizia ad acquisire dati dal digitizer. Di default nessun thread di trattamento dei dati e' attivo.
        output_module->Output("Data acquisition starting...\n");

        miodigitizer.DigitizerStateMachineStartReading ();

      }

    //Se e' stato ricevuto il comando stop.
    else if (DigitizerFlowControl::
         reg_matches (buffer, "^[sS][tT][oO][pP][ \t]*$")
         || command_received_by_user.command_sent_by_user == STOP)
      {
        //Il programma termina l'acquisizione dei dati dal digitizer.
        output_module->Output("Stopping data acquisition...\n");

        miodigitizer.DigitizerStateMachineStopReading ();

      }

    //Se e' stato ricevuto il comando prestart.
    else if (DigitizerFlowControl::
         reg_matches (buffer, "^[pP][rR][eE][sS][tT][aA][rR][tT][ \t]*$")
         || command_received_by_user.command_sent_by_user ==
         PRESTART)
      {
        //Il programma inizia il preprocessamento dei dati raccolti.
        output_module->Output("Starting Thread Preprocessing...\n");
        miodigitizer.DigitizerStateStartPreprocessing ();
      }

    //Se e' stato ricevuto il comando prestop.
    else if (DigitizerFlowControl::
         reg_matches (buffer, "^[pP][Rr][Ee][Ss][Tt][Oo][Pp][ \t]*$")
         || command_received_by_user.command_sent_by_user ==
         PRESTOP)
      {
        //Il programma termina il preprocessamento dei dati raccolti.
        output_module->Output("Stopping Thread Preprocessing...\n");
        miodigitizer.DigitizerStateStopPreprocessing ();
      }

    //Se e' stato ricevuto il comando rawstart.
    else if (DigitizerFlowControl::
         reg_matches (buffer, "^[rR][aA][wW][sS][tT][aA][rR][tT][ \t]*$")
         || command_received_by_user.command_sent_by_user ==
         RAWSTART)
      {
        //Il programma inizia la scrittura su disco dei dati raccolti.
        output_module->Output("Starting Thread RawDataWriting...\n");
        miodigitizer.DigitizerStateStartRawDataWriting ();
      }

    //Se e' stato ricevuto il comando rawstop.
    else if (DigitizerFlowControl::
         reg_matches (buffer, "^[Rr][Aa][Ww][Ss][Tt][Oo][Pp][ \t]*$")
         || command_received_by_user.command_sent_by_user ==
         RAWSTOP)
      {
        //Il programma termina la scrittura su disco dei dati raccolti.
        output_module->Output("Stopping Thread RawDataWriting...\n");
        miodigitizer.DigitizerStateStopRawDataWriting ();
      }

    //Se e' stato ricevuto il comando vistart.
    else if (DigitizerFlowControl::
         reg_matches (buffer, "^[Vv][Ii][Ss][Tt][Aa][Rr][Tt][ \t]+[0-9]{1,3}[ \t]*$"))
      {
            digitizer_flow_control_application_setup -> channel_visualized = FindIntegerValue(buffer);
        //Il programma inizia la visualizzazione dei dati raccolti.
        output_module->Output("Starting Thread Visualization...\n");
        miodigitizer.DigitizerStateStartVisualization ();
      }

    //Se e' stato ricevuto il comando vistop.
    else if (DigitizerFlowControl::
         reg_matches (buffer, "^[Vv][Ii][Ss][Tt][Oo][Pp][ \t]*$"))
      {
        //Il programma termina la visualizzazione dei dati raccolti.
        output_module->Output("Stopping Thread Visualization...\n");
        miodigitizer.DigitizerStateStopVisualization ();
      }

    //Se e' stato ricevuto il comando send.
    else if (DigitizerFlowControl::
         reg_matches (buffer, "^[Ss][Ee][Nn][Dd][ \t]*$")
         || command_received_by_user.command_sent_by_user == SEND)
      {
        //Il programma invia un software trigger
        output_module->Output("Sending software trigger...\n");
        miodigitizer.DigitizerStateMachineSendSWTrigger ();
      }

    //Se e' stato ricevuto il comando close
    else if (DigitizerFlowControl::
         reg_matches (buffer, "^[cC][lL][oO][sS][eE][ \t]*$")
         || command_received_by_user.command_sent_by_user == CLOSE)
      {
        //Il programma chiude il digitizer
        output_module->Output("Closing Digitizer...\n");
        miodigitizer.DigitizerStateMachineQuit ();
      }

    //Se e' stato ricevuto il comando print.
    else if (DigitizerFlowControl::
         reg_matches (buffer, "^[pP][rR][iI][nN][tT][ \t]*$")
         || command_received_by_user.command_sent_by_user == PRINT)
      {
        //Il programma stampa in formato comprensibile le informazioni contenuto nell'oggetto di tipo ConfObject che viene usato per configurare il
        //digitizer. Le informazioni contenuto nell'oggetto sono prese dal file di configurazione con i comandi init (attributo open) e setup (tutti gli
        //altri).
        output_module->Output("Printing ConfObject parameters in human readable format...\n");
        miodigitizer.digitizer.internal_config.PrintAllHuman ();
      }

    //Se e' stato ricevuto il comando print files.
    else if (DigitizerFlowControl::
         reg_matches (buffer,
              "^[pP][rR][iI][nN][tT][ \t]+[Ff][Ii][Ll][Ee][Ss][ \t]*$")
         || command_received_by_user.command_sent_by_user ==
         PRINTFILES)
      {
        //Viene stampato il path dei file di configurazione, di log e di salvataggio dei rawdata.

        //Path del file di configurazione.
        if (digitizer_flow_control_application_setup->
        application_setup_conf_file_path != NULL)
      {
        output_module->Output(digitizer_flow_control_application_setup->
            application_setup_conf_file_path);
        output_module->Output("\n");
      }

        //Path del file di salvataggio dei rawdata.
        if (digitizer_flow_control_application_setup->
        application_setup_data_file_path != NULL)
      {
        output_module->Output(digitizer_flow_control_application_setup
->
            application_setup_data_file_path);
        output_module->Output("\n");
      }

        //Path del file di log.
        if (digitizer_flow_control_application_setup->
        application_setup_log_file_path != NULL)
      {
        output_module->Output(digitizer_flow_control_application_setup
->
            application_setup_log_file_path);
        output_module->Output("\n");
      }
      }

    //Se e' stato ricevuto il comando help (solo da stdin perche' i clients connessi via tcp hanno il
loro comando help).
    else if (DigitizerFlowControl::
         reg_matches (buffer, "^[hH][eE][lL][pP][ \t]*$"))
      {
        //Viene stampata la lista dei comandi disponibili.
        printf ("Listing commands available...\n");
        DigitizerFlowControl::Help ();
      }

    //Se e' stato ricevuto il comando -f [pathdellogfile]
    else if (DigitizerFlowControl::
         reg_matches (buffer, "^[-][Ff][ ].+$")
         || command_received_by_user.command_sent_by_user ==
         CHANGECONF)
      {
        //Se il comando e' stato inviato da tcp, viene ricavato il path del file di configurazione dalla
stringa contenuta in buffer.
        if (command_received_by_user.command_sent_by_user ==
CHANGECONF)
      {
        output_module->Output("Changing conf file...\n");
        free ((void *) digitizer_flow_control_application_setup->
          application_setup_conf_file_path);
        digitizer_flow_control_application_setup->
          application_setup_conf_file_path = NULL;
        digitizer_flow_control_application_setup->
          application_setup_conf_file_path =
          (char *)
          malloc (strlen (command_received_by_user.first_parameter)
              + 1);
        strcpy ((char *) digitizer_flow_control_application_setup
->
            application_setup_conf_file_path,
            command_received_by_user.first_parameter);
      }
        else
      {
        //Se il comando e' stato inviato da stdin, viene ricavato il path del file di configurazione
dalla stringa contenuta in buffer.
        //In questo caso la stringa e' un po' diversa rispetto a quella del caso precedente: per questo
e' necessario distinguere le procedure
        //di ottenimento della stringa.
        my_punt = FindPointer (buffer);
        output_module->Output("Changing conf file...\n");
        free ((void *) digitizer_flow_control_application_setup->
          application_setup_conf_file_path);
        digitizer_flow_control_application_setup->
          application_setup_conf_file_path = NULL;
        digitizer_flow_control_application_setup->
          application_setup_conf_file_path =
          (char *) malloc (strlen (my_punt) + 1);
        strcpy ((char *) digitizer_flow_control_application_setup
->
            application_setup_conf_file_path, my_punt);
      }
      }

    //Se e' stato ricevuto il comando -f [pathdelrawdatafile]
    else if (DigitizerFlowControl::
         reg_matches (buffer, "^[-][Dd][ ].+$")
         || command_received_by_user.command_sent_by_user ==
         CHANGEDATA)
      {
        //WARNING: modificare il path del data file non e' banale perche' si richia l'inconsistenza delle
operazioni se l'acquisizione e' attiva.
        //Qui si e' preferito interrompere l'acquisizione, modificare il path del data file e riprendere
l'acquisizione.
        //Se il comando e' stato inviato da tcp, viene ricavato il path del file di salvataggio dei
rawdata dalla stringa contenuta in buffer.
```

```
          if (command_received_by_user.command_sent_by_user == CHANGEDATA)
        {
          output_module->Output ("Changing data file...\n");
          if (miodigitizer.go_raw_data == 1)
            {
              support_flag1 = 1;
              miodigitizer.DigitizerStateStopRawDataWriting ();
            }
          digitizer_flow_control_application_setup->
            ApplicationSetupDataFileModify (command_received_by_user.
                            first_parameter);
          if (support_flag1 == 1)
            {
              support_flag1 = 0;
              miodigitizer.DigitizerStateStartRawDataWriting ();
            }
        }
          else
        {
          //Se il comando e' stato inviato da stdin, viene ricavato il path del file di salvataggio dei rawdata dalla stringa contenuta in buffer.
          //In questo caso la stringa e' un po' diversa rispetto a quella del caso precedente: per questo e' necessario distinguere le procedure
          //di ricavo della stringa.
          my_punt = FindPointer (buffer);
          output_module->Output ("Changing data file...\n");
          if (miodigitizer.go_raw_data == 1)
            {
              support_flag1 = 1;
              miodigitizer.DigitizerStateStopRawDataWriting ();
            }
          digitizer_flow_control_application_setup->
            ApplicationSetupDataFileModify (my_punt);
          if (support_flag1 == 1)
            {
              support_flag1 = 0;
              miodigitizer.DigitizerStateStartRawDataWriting ();
            }
        }
        }

      //Se e' stato ricevuto il comando -l [pathdellogfile]
      else if (DigitizerFlowControl::
           reg_matches (buffer, "^[-][Ll][ ].+$")
           || command_received_by_user.command_sent_by_user ==
           CHANGELOG)
        {
          //Se il comando e' stato inviato da tcp, viene ricavato il path del log file dalla stringa contenuta in buffer.
          if (command_received_by_user.command_sent_by_user == CHANGELOG)
        {
          output_module->Output ("Changing log file...\n");
          free ((void *) digitizer_flow_control_application_setup->
            application_setup_log_file_path);
          digitizer_flow_control_application_setup->
            application_setup_log_file_path = NULL;
          digitizer_flow_control_application_setup->
            application_setup_log_file_path =
            (char *)
            malloc (strlen (command_received_by_user.first_parameter)
                + 1);
          strcpy ((char *) digitizer_flow_control_application_setup->
              application_setup_log_file_path,
              command_received_by_user.first_parameter);
          logfile->
            LogFileSet (digitizer_flow_control_application_setup->
                application_setup_log_file_path);
        }
          else
        {
          //Se il comando e' stato inviato da stdin, viene ricavato il path del log file dalla stringa contenuta in buffer.
          //In questo caso la stringa e' un po' diversa rispetto a quella del caso precedente: per questo e' necessario distinguere le procedure
          //di ricavo della stringa.
          my_punt = FindPointer (buffer);
          output_module->Output ("Changing log file...\n");
          free ((void *) digitizer_flow_control_application_setup->
            application_setup_log_file_path);
          digitizer_flow_control_application_setup->
            application_setup_log_file_path = NULL;
          digitizer_flow_control_application_setup->
            application_setup_log_file_path =
            (char *) malloc (strlen (my_punt) + 1);
          strcpy ((char *) digitizer_flow_control_application_setup
->
              application_setup_log_file_path, my_punt);
          logfile->
            LogFileSet (digitizer_flow_control_application_setup->
                application_setup_log_file_path);
        }
        }

      //Se e' stato inviato il comando check.
      else if (DigitizerFlowControl::
           reg_matches (buffer, "^[Cc][Hh][Ee][Cc][Kk][ \t]*$")
           || command_received_by_user.command_sent_by_user ==
CHECK)
        {
          //Viene controllato se il file di configurazione contiene manca dei parametri fondamentali per
 impostare il digitizer.
          output_module->Output("Checking configuration file...\n");
          my_consistence.
        ConfigurationConsistenceConfFileSetupEssentialWithInitCheck
        (digitizer_flow_control_application_setup->
         application_setup_conf_file_path);
        }

      //Se e' stato inviato il comando chkconf.
      else if (DigitizerFlowControl::
           reg_matches (buffer, "^[Cc][Hh][Kk][Cc][Oo][Nn][Ff][ \t]*$")
           || command_received_by_user.command_sent_by_user ==
           CHKCONF)
        {
          //Viene stampato il contenuto del file di configurazione: come contenuto si intendono le stringhe
 riconosciute dallo scanner generato in flex.
          output_module->Output("These are the settings read from the configuration file...\n");
          AnalizzaPrint (digitizer_flow_control_application_setup
->
                 application_setup_conf_file_path);
        }

      //Se e' stato inviato il comando write register [address] [data].
      else if (DigitizerFlowControl::
           reg_matches (buffer,
                "^[Ww][Rr][Ii][Tt][Ee][ \t]+[Rr][Ee][Gg][Ii][Ss][Tt][Ee][Rr][ \t]+(0x[0-9a-fA-F]{1,16})[ \t
]+(0x[0-9a-fA-F]{1,16})[ \t]*$")
           || command_received_by_user.command_sent_by_user ==
           WRITEREGISTER)
        {

          Data = -1;

          //Se il comando e' stato inviato da tcp, vengono ricavati i dati dalla stringa inviata.

          if (command_received_by_user.command_sent_by_user ==
          WRITEREGISTER)
        {
          Address =
            strtoul (command_received_by_user.first_parameter, NULL,
                 16);
          my_punt =
            FindPointer (command_received_by_user.first_parameter);
          Data = strtoul (my_punt, NULL, 16);
          bzero (support_string, STANDARDBUFFERLIMIT);
          snprintf (support_string, STANDARDBUFFERLIMIT,
                "Writing 0x%x in the 0x%x register\n", Data,
                Address);
          output_module->Output(support_string);
          miodigitizer.digitizer.
            DigitizerObjectWriteRegister (Address, Data);

          //Viene stampato il contenuto del registro modificato per poter rendersi subito conto se la
 modifica e' avvenuto con successo.
          Data = -1;
          miodigitizer.digitizer.
            DigitizerObjectReadRegister (Address, &Data);
          bzero (support_string, STANDARDBUFFERLIMIT);
          snprintf (support_string, STANDARDBUFFERLIMIT,
                "The data in the 0x%x register are: 0x%x\n",
                Address, Data);
          output_module->Output(support_string);
        }
          else
        {
          //Se il comando e' stato inviato da stdin, vengono ricavati i dati dalla stringa inviata: la
 stringa e' leggermente diversa rispetto a quella del
          //caso precedente quindi e' stato necessario distinguere i due casi.
          my_punt = FindPointer (buffer);
          my_punt = FindPointer (my_punt);
          Address = strtoul (my_punt, NULL, 16);
```

```
    my_punt = FindPointer (my_punt);
    Data = strtoul (my_punt, NULL, 16);
    bzero (support_string, STANDARDBUFFERLIMIT);
    snprintf (support_string, STANDARDBUFFERLIMIT,
          "Writing 0x%x in the 0x%x register\n", Data,
          Address);
    output_module->Output(support_string);
    miodigitizer.digitizer.
      DigitizerObjectWriteRegister (Address, Data);

    //Viene stampato il contenuto del registro modificato per poter rendersi subito conto se la modifica e' avvenuto con successo.
    Data = -1;
    miodigitizer.digitizer.
      DigitizerObjectReadRegister (Address, &Data);
    bzero (support_string, STANDARDBUFFERLIMIT);
    snprintf (support_string, STANDARDBUFFERLIMIT,
          "The data in the 0x%x register are: 0x%x\n",
          Address, Data);
    output_module->Output(support_string);
  }
  }

//Se e' stato inviato il comando read register [address] [data].
else if (DigitizerFlowControl::
     reg_matches (buffer,
          "^[Rr][Ee][Aa][Dd][ \t]+[Rr][Ee][Gg][Ii][Ss][Tt][Ee][Rr][ \t]+(0x[0-9a-fA-F]{1,16})[ \t]*$")
     || command_received_by_user.command_sent_by_user ==
     READREGISTER)
  {

    Data = -1;

    //Se l'input e' stato ricevuto via tcp.
    if (command_received_by_user.command_sent_by_user ==
    READREGISTER)
  {
    //Ricavo dalla stringa contenuta in buffer l'indirizzo del registro da leggere.

    Address =
      strtoul (command_received_by_user.first_parameter, NULL,
           16);
    miodigitizer.digitizer.
      DigitizerObjectReadRegister (Address, &Data);
    bzero (support_string, STANDARDBUFFERLIMIT);
    snprintf (support_string, STANDARDBUFFERLIMIT,
          "I've found this data in the 0x%x register: 0x%x\n",
          Address, Data);
    output_module->Output(support_string);
  }
    else
  {
    //Se l'input e' stato ricevuto via stdin, ricavo dalla stringa contenuta in buffer l'indirizzo del registro da leggere.
    my_punt = FindPointer (buffer);
    my_punt = FindPointer (my_punt);
    Address = strtoul (my_punt, NULL, 16);
    miodigitizer.digitizer.
      DigitizerObjectReadRegister (Address, &Data);
    bzero (support_string, STANDARDBUFFERLIMIT);
    snprintf (support_string, STANDARDBUFFERLIMIT,
          "I've found this data in the 0x%x register: 0x%x\n",
          Address, Data);
    output_module->Output(support_string);
  }
  }

//Se e' stato inviato il comando "more".
else if (DigitizerFlowControl::
     reg_matches (buffer, "^[Mm][Oo][Rr][Ee][ \t]*$")
     || command_received_by_user.command_sent_by_user ==
MORE)
  {
    //Viene letto il contenuto del log file.
    logfile->LogFileRead ();

  }

//Se e' stato ricevuto il comando status.
else if (DigitizerFlowControl::
     reg_matches (buffer, "^[Ss][Tt][Aa][Tt][Uu][Ss][ \t]*$")
     || command_received_by_user.command_sent_by_user ==
STATUS)
  {
    //Viene stampato lo stato (ON/OFF) del thread che acquisisce i dati e di quelli che operano sui dati
```

```
          miodigitizer.DigitizerStateMachinePrintStatus ();
        }

      //Se il comando non e' stato riconosciuto.
      else
        {
          output_module->StdOutInsert ("Unrecognized command\n");
          output_module->StdOutPrint ();
        }

    }
      else
    pthread_cond_wait (&input_flow_cond, &input_flow_mutex);
      // strlen (buffer) != 0 || command_received_by_user.command_sent_by_user != 0
      pthread_mutex_unlock (&input_flow_mutex);
    }   // while 1

}   //void DigitizerFlowControl::DigitizerFlowControlStart ()

//Funzione per confrontare una stringa con un'espressione regolare.
bool DigitizerFlowControl::reg_matches (const char *str, const char *
pattern)
{
  regex_t
    re;
  int
    ret;

  if (regcomp (&re, pattern, REG_EXTENDED) != 0)
    return false;

  ret = regexec (&re, str, (size_t) 0, NULL, 0);
  regfree (&re);

  if (ret == 0)
    return true;

  return false;
}

//Funzione di aiuto.
void
DigitizerFlowControl::Help ()
{
  printf ("Available command list:\n");
  printf ("init: open the digitizer\n");
  printf ("setup: setup the digitizer\n");
  printf ("start: start the data acquisition\n");
  printf ("stop: stop the data acquisition\n");
  printf ("prestart: start the preprocessing thread\n");
  printf ("prestop: stop the preprocessing thread\n");
  printf ("vistart [channelnumber]: start the visualization thread\n");
  printf ("vistop: stop the visualization thread\n");
  printf ("rawstart: start the raw data writing thread\n");
  printf ("rawstop: stop the raw data writing thread\n");
  printf ("close: close the digitizer\n");
  printf ("send: send a software trigger\n");
  printf ("help\n");
  printf ("check: check the correctness of the configuration file\n");
  printf ("chkconf: print the content of the configuration file\n");
  printf ("write register 0x[register] 0x[data]\n");
  printf ("read register 0x[register]\n");
  printf ("-f [conf file path]: change the configuration file path\n");
  printf ("-d [data file path]: change the data file path\n");
  printf ("-l [log file path]: change the log file path\n");
  printf ("print: print the internal configuration object used to configure the digitizer\n");
  printf ("print files: print the path of the configuration file, of the data file and of the log file\n");
  printf ("status: print the status of the threads acquisition, preprocessing, raw data and visualization\n
");
  printf ("more: display the content of the logfile\n");
  printf ("exit: quit program\n");
  printf ("quit: quit program\n");
}

//Funzione per stampare il significato dei comandi inviati via tcp.
const char *
DigitizerFlowControl::ParseCommand (int comando_inviato_da_tcp)
{
  switch (comando_inviato_da_tcp)
    {

    case INIT:

      return "init";

    case SETUP:

      return "setup";

    case PRESTART:

      return "prestart";

    case PRESTOP:

      return "prestop";

    case VISTART:

      return "vistart";

    case VISTOP:

      return "vistop";

    case START:

      return "start";

    case STOP:

      return "stop";

    case SEND:

      return "send";

    case CLOSE:

      return "close";

    case QUIT:

      return "quit";

    case RAWSTART:

      return "rawstart";

    case RAWSTOP:

      return "rawstop";

    case PRINT:

      return "print";

    case CHECK:

      return "check";

    case CHKCONF:

      return "chkconf";

    case MORE:

      return "more";

    case CHANGECONF:

      return "-f [conf_file_path]";

    case CHANGEDATA:

      return "-d [data_file_path]";

    case CHANGELOG:

      return "-l [log_file_path]";

    case WRITEREGISTER:

      return "write register";

    case READREGISTER:

      return "read register";

    case PRINTFILES:

      return "print files";

    case STATUS:

      return "status";

    default:
      return "Unrecognized command";
    }
}
```

6.35 DigitizerFlowControl.h File Reference

Classes

- class DigitizerFlowControl

  The DigitizerFlowControl class controls the flow of execution of the program.

6.36 DigitizerFlowControl.h

```

class DigitizerFlowControl
{
public:

  static pthread_cond_t input_flow_cond;

  static pthread_mutex_t input_flow_mutex;

  Input mioinput;

  CommunicationObject mioTCP;

  ApplicationSetup *digitizer_flow_control_application_setup;

    DigitizerFlowControl ();

  void DigitizerFlowControlStart ();

  const char *ParseCommand (int recvline);

  void Help ();

  bool reg_matches (const char *str, const char *pattern);
};
```

6.37 DigitizerObject.cpp File Reference

```
#include "DefineGeneral.h"
#include "ConfObject.h"
#include "DigitizerErrorObject.h"
#include "LogFile.h"
#include "DigitizerObject.h"
#include "Analizzatore.h"
#include "ApplicationSetup.h"
#include "OutputModule.h"
#include <assert.h>
#include <stdio.h>
#include <CAENDigitizer.h>
```

6.37.1 Detailed Description

Author

Daniele Berto

Definition in file DigitizerObject.cpp.

6.38 DigitizerObject.cpp

```
#include "DefineGeneral.h"
#include "ConfObject.h"
#include "DigitizerErrorObject.h"
#include "LogFile.h"
#include "DigitizerObject.h"
#include "Analizzatore.h"
#include "ApplicationSetup.h"
#include "OutputModule.h"
#include <assert.h>
#include <stdio.h>
#include <CAENDigitizer.h>

DigitizerObject::DigitizerObject (const char *config_file)
{
  if (AnalizzaInit (&internal_config, config_file))
    logfile->
      LogFileWriteString
      ("Warning: can't create DigitizerObject correctly. Configuration file does't exists\n");
  if (AnalizzaSetup (&internal_config, config_file))
    logfile->
      LogFileWriteString
      ("Warning: can't create DigitizerObject correctly. Configuration file does't exists\n");
  set_board_info = 0;
  logfile = LogFile::Instance ();
}

DigitizerObject::DigitizerObject (ConfObject config)
{
  internal_config = config;
  set_board_info = 0;
  logfile = LogFile::Instance ();
}

void
DigitizerObject::
DigitizerObjectSetConfigStructureConfObject (
ConfObject config)
{
  internal_config = config;
  logfile = LogFile::Instance ();
}

DigitizerObject::DigitizerObject ()
{
  set_board_info = 0;
  logfile = LogFile::Instance ();
}

void
DigitizerObject::
DigitizerObjectSetConfigStructureInit (const char *config_file)
{
  if (AnalizzaInit (&internal_config, config_file))
    logfile->
      LogFileWriteString
      ("Warning: can't create DigitizerObject correctly. Configuration file does't exists\n");
}

void
DigitizerObject::
DigitizerObjectSetConfigStructureSetup (const char *config_file)
{
  if (AnalizzaSetup (&internal_config, config_file))
    logfile->
      LogFileWriteString
      ("Warning: can't create DigitizerObject correctly. Configuration file does't exists\n");
}

int
DigitizerObject::DigitizerObjectOpen ()
{

  char stringa[STANDARDBUFFERLIMIT];
  bzero (stringa, STANDARDBUFFERLIMIT);

  CAEN_DGTZ_ConnectionType Connection;

  if (internal_config.LinkType == 0)
    Connection = CAEN_DGTZ_USB;
  else if (internal_config.LinkType == 1)
    Connection = CAEN_DGTZ_OpticalLink;
  else
    {
      snprintf (stringa, STANDARDBUFFERLIMIT,
        "%s %d Configuration structure invalid: LinkType %d field inconsistent\n",
        __FILE__, __LINE__, internal_config.LinkType);
      logfile->LogFileWriteString (stringa);
      return 1;
    }

  ret =
    CAEN_DGTZ_OpenDigitizer (Connection, internal_config.LinkNumber,
                 internal_config.ConetNode,
                 internal_config.VMEBaseAddress, &handle);
  //ret_error.DigitizerErrorObjectDebugging(ret, __FILE__, __func__, __LINE__);
  logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
  //ret_error.DigitizerErrorObjectPrintError (ret);

  return 0;
}

int
DigitizerObject::DigitizerObjectReset ()
{
  ret = CAEN_DGTZ_Reset (handle);
  //ret_error.digitizer_error_object_debugging(ret, __FILE__, __LINE__);
  //ret_error.DigitizerErrorObjectPrintError (ret);
  //ret_error.DigitizerErrorObjectDebugging(ret, __FILE__, __func__, __LINE__);
  logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
  return 0;
}

int
DigitizerObject::DigitizerObjectGetInfo ()
{
  ret = CAEN_DGTZ_GetInfo (handle, &BoardInfo);
  //ret_error.digitizer_error_object_debugging(ret, __FILE__, __LINE__);
  //ret_error.DigitizerErrorObjectPrintError (ret);
  //ret_error.DigitizerErrorObjectDebugging(ret, __FILE__, __func__, __LINE__);
  logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
  if (ret == CAEN_DGTZ_Success)
    set_board_info = 1;
  return 0;
}

int
DigitizerObject::GetFamilyCode(int * FamilyCode)
{

char stringa[STANDARDBUFFERLIMIT];

  if (set_board_info == 0)
    {
      bzero (stringa, STANDARDBUFFERLIMIT);
      snprintf (stringa, STANDARDBUFFERLIMIT,
        "File: %s, Function %s, Line: %d, Error: can't print board info. Use DigitizerObject::DigitizerObjectgetinfo first. Perhaps you have not open the digitizer?\n",
        __FILE__, __func__, __LINE__);
      logfile->LogFileWriteString (stringa);
      return 1;
    }

*FamilyCode = BoardInfo.FamilyCode;

}

int
DigitizerObject::GetFormFactorCode(int * FormFactor)
{

char stringa[STANDARDBUFFERLIMIT];

  if (set_board_info == 0)
    {
      bzero (stringa, STANDARDBUFFERLIMIT);
      snprintf (stringa, STANDARDBUFFERLIMIT,
        "File: %s, Function %s, Line: %d, Error: can't print board info. Use DigitizerObject::DigitizerObjectgetinfo first. Perhaps you have not open the digitizer?\n",
        __FILE__, __func__, __LINE__);
      logfile->LogFileWriteString (stringa);
      return 1;
    }

*FormFactor = BoardInfo.FormFactor;

}

void
DigitizerObject::PrintBoardInfo ()
{
  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  char stringa[STANDARDBUFFERLIMIT];

  if (set_board_info == 0)
    {
      bzero (stringa, STANDARDBUFFERLIMIT);
      snprintf (stringa, STANDARDBUFFERLIMIT,
        "File: %s, Function %s, Line: %d, Error: can't print board info. Use
 DigitizerObject::DigitizerObjectgetinfo first. Perhaps you have not open the digitizer?\n",
        __FILE__, __func__, __LINE__);
      logfile->LogFileWriteString (stringa);
      return;
    }

  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT, "Board model: %s\n",
        BoardInfo.ModelName);
  output_module->Output(stringa);

  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT, "Model: %d\n",
BoardInfo.Model);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT, "Channels: %d\n",
        BoardInfo.Channels);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT, "Form factor: %d\n",
        BoardInfo.FormFactor);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT, "FamilyCode: %d\n",
        BoardInfo.FamilyCode);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT, "ROC_FirmwareRel: %s\n",
        BoardInfo.ROC_FirmwareRel);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT, "AMC_FirmwareRel: %s\n",
        BoardInfo.AMC_FirmwareRel);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT, "SerialNumber: %d\n",
        BoardInfo.SerialNumber);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT, "PCB_Revision: %d\n",
        BoardInfo.PCB_Revision);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT, "ADC_NBits: %d\n",
        BoardInfo.ADC_NBits);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT, "SAMCorrectionDataLoaded: %d\n",
        BoardInfo.SAMCorrectionDataLoaded);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT, "CommHandle: %d\n",
        BoardInfo.CommHandle);
  output_module->Output(stringa);


  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf (stringa, STANDARDBUFFERLIMIT, "License: %s\n",
BoardInfo.License);
  output_module->Output(stringa);


}

int
DigitizerObject::DigitizerObjectSetRecordLength (int
recordlength)
{

  ret = CAEN_DGTZ_SetRecordLength (handle, recordlength);
  //ret_error.DigitizerErrorObjectPrintError (ret);
  //ret_error.DigitizerErrorObjectDebugging(ret, __FILE__, __func__, __LINE__);
  logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
  return 0;

}

int
DigitizerObject::DigitizerObjectSetGroupEnableMask (int
enablemask)
{
  ret = CAEN_DGTZ_SetGroupEnableMask (handle, enablemask);
  //ret_error.DigitizerErrorObjectPrintError (ret);
  //ret_error.DigitizerErrorObjectDebugging(ret, __FILE__, __func__, __LINE__);
  logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
  return 0;
}

int
DigitizerObject::DigitizerObjectWriteRegister (int registry,
int data)
{

  ret = CAEN_DGTZ_WriteRegister (handle, registry, data);
  //ret_error.DigitizerErrorObjectPrintError (ret);
  //ret_error.DigitizerErrorObjectDebugging(ret, __FILE__, __func__, __LINE__);
  logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
  return 0;

}

int
DigitizerObject::DigitizerObjectReadRegister (int registry, int
 *data)
{

  ret =
    CAEN_DGTZ_ReadRegister (handle, (uint32_t) registry, (uint32_t *) data);
  //ret_error.DigitizerErrorObjectPrintError (ret);
  //ret_error.DigitizerErrorObjectDebugging(ret, __FILE__, __func__, __LINE__);
  logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
  return 0;

}

int
DigitizerObject::DigitizerObjectGetDRS4SamplingFrequency
  (CAEN_DGTZ_DRS4Frequency_t * frequenza)
{

  ret = CAEN_DGTZ_GetDRS4SamplingFrequency (handle, frequenza);
  //ret_error.DigitizerErrorObjectPrintError (ret);
  //ret_error.DigitizerErrorObjectDebugging(ret, __FILE__, __func__, __LINE__);
  logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
  return 0;

}

int
DigitizerObject::DigitizerObjectSetDRS4SamplingFrequency
  (CAEN_DGTZ_DRS4Frequency_t frequenza)
```

```
{

  ret = CAEN_DGTZ_SetDRS4SamplingFrequency (handle, frequenza);
  //ret_error.DigitizerErrorObjectPrintError (ret);
  //ret_error.DigitizerErrorObjectDebugging(ret, __FILE__, __func__, __LINE__);
  logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
  return 0;

}

int
DigitizerObject::DigitizerObjectSetChannelDCOffset (int channel, int dc_mask)
{

  ret = CAEN_DGTZ_SetChannelDCOffset (handle, channel, dc_mask);
  //ret_error.DigitizerErrorObjectPrintError (ret);
  //ret_error.DigitizerErrorObjectDebugging(ret, __FILE__, __func__, __LINE__);
  logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
  return 0;

}

int
DigitizerObject::DigitizerObjectSetGroupDCOffset (int group, int dc_mask)
{

  ret = CAEN_DGTZ_SetGroupDCOffset (handle, group, dc_mask);
  ret_error.DigitizerErrorObjectPrintError (ret);
  return 0;

}

int
DigitizerObject::DigitizerObjectSetMaxNumEventsBLT (int MaxNumEventsBLT)
{

  ret = CAEN_DGTZ_SetMaxNumEventsBLT (handle, MaxNumEventsBLT);
  //ret_error.DigitizerErrorObjectPrintError (ret);
  //ret_error.DigitizerErrorObjectDebugging(ret, __FILE__, __func__, __LINE__);
  logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
  return 0;

}

int
DigitizerObject::DigitizerObjectSetAcquisitionMode (CAEN_DGTZ_AcqMode_t
                            AcqMode)
{

  ret = CAEN_DGTZ_SetAcquisitionMode (handle, AcqMode);
  //ret_error.DigitizerErrorObjectPrintError (ret);
  //ret_error.DigitizerErrorObjectDebugging(ret, __FILE__, __func__, __LINE__);
  logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
  return 0;

}

int
DigitizerObject::DigitizerObjectClose ()
{

  ret = CAEN_DGTZ_CloseDigitizer (handle);
  //ret_error.DigitizerErrorObjectPrintError (ret);
  //ret_error.DigitizerErrorObjectDebugging(ret, __FILE__, __func__, __LINE__);
  logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
  return 0;

}

int
DigitizerObject::DigitizerObjectSetExtTriggerInputMode
  (CAEN_DGTZ_TriggerMode_t TriggerMode)
{

  ret = CAEN_DGTZ_SetExtTriggerInputMode (handle, TriggerMode);
  //ret_error.DigitizerErrorObjectPrintError (ret);
  //ret_error.DigitizerErrorObjectDebugging(ret, __FILE__, __func__, __LINE__);
  logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
  return 0;

}

```

```
int
DigitizerObject::DigitizerObjectSetSWTriggerMode (
CAEN_DGTZ_TriggerMode_t
                          TriggerMode)
{

  ret = CAEN_DGTZ_SetSWTriggerMode (handle, TriggerMode);
  //ret_error.DigitizerErrorObjectPrintError (ret);
  //ret_error.DigitizerErrorObjectDebugging(ret, __FILE__, __func__, __LINE__);
  logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
  return 0;

}

int
DigitizerObject::DigitizerObjectSWStartAcquisition ()
{

  ret = CAEN_DGTZ_SWStartAcquisition (handle);
  //ret_error.DigitizerErrorObjectPrintError (ret);
  //ret_error.DigitizerErrorObjectDebugging(ret, __FILE__, __func__, __LINE__);
  logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
  return 0;

}

int
DigitizerObject::DigitizerObjectSWStopAcquisition ()
{

  ret = CAEN_DGTZ_SWStopAcquisition (handle);
  //ret_error.DigitizerErrorObjectPrintError (ret);
  //ret_error.DigitizerErrorObjectDebugging(ret, __FILE__, __func__, __LINE__);
  logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
  return 0;

}
```

6.39 DigitizerObject.h File Reference

```
#include <assert.h>
#include <stdio.h>
#include <CAENDigitizer.h>
```

Classes

- class DigitizerObject

  The DigitizerObject class envelops CAEN_DGTZ functions from CAENDigitizer library.

6.39.1 Detailed Description

Author

Daniele Berto

Definition in file DigitizerObject.h.

6.40 DigitizerObject.h

```

#include <assert.h>
#include <stdio.h>
#include <CAENDigitizer.h>

class DigitizerObject
{

```

```
public:

  CAEN_DGTZ_ErrorCode ret;

  DigitizerErrorObject ret_error;

  ConfObject internal_config;

  int set_board_info;

  int handle;

  CAEN_DGTZ_BoardInfo_t BoardInfo;

    DigitizerObject ();

    DigitizerObject (const char *config_file);

    DigitizerObject (ConfObject config);

  void DigitizerObjectSetConfigStructureConfObject (
ConfObject config);

  void DigitizerObjectSetConfigStructureInit (const char *config_file)
;

  void DigitizerObjectSetConfigStructureSetup (const char *
config_file);

  int DigitizerObjectOpen ();

  int DigitizerObjectClose ();

  int DigitizerObjectReset ();

  int DigitizerObjectGetInfo ();

  CAEN_DGTZ_BoardInfo_t GetBoardInfo ();

  int GetFamilyCode(int * FamilyCode);

  int GetFormFactorCode(int * FormFactor);

  void PrintBoardInfo ();

  int DigitizerObjectSetRecordLength (int recordlength);

  int DigitizerObjectSetGroupEnableMask (int enablemask);

  int DigitizerObjectWriteRegister (int registry, int data);

  int DigitizerObjectReadRegister (int registry, int *data);

  int DigitizerObjectGetDRS4SamplingFrequency (
CAEN_DGTZ_DRS4Frequency_t *
                            frequenza);

  int DigitizerObjectSetDRS4SamplingFrequency (
CAEN_DGTZ_DRS4Frequency_t
                            frequenza);

  int DigitizerObjectSetChannelDCOffset (int channel_mask, int dc_mask);

  int DigitizerObjectSetGroupDCOffset (int group_mask, int dc_mask);

  int DigitizerObjectSetMaxNumEventsBLT (int MaxNumEventsBLT);

  int DigitizerObjectSetAcquisitionMode (CAEN_DGTZ_AcqMode_t AcqMode);

  int DigitizerObjectSetExtTriggerInputMode (CAEN_DGTZ_TriggerMode_t
                          TriggerMode);

  int DigitizerObjectSetSWTriggerMode (CAEN_DGTZ_TriggerMode_t TriggerMode);

  int DigitizerObjectSWStartAcquisition ();

  int DigitizerObjectSWStopAcquisition ();

  LogFile *logfile;
};
```

6.41 DigitizerObjectGeneric.cpp File Reference

```
#include "ConfObject.h"
#include "DigitizerErrorObject.h"
#include "LogFile.h"
#include "DigitizerObject.h"
#include "DigitizerObjectGeneric.h"
#include "Analizzatore.h"
#include "DefineGeneral.h"
#include <assert.h>
#include <stdio.h>
#include <CAENDigitizer.h>
```

6.42 DigitizerObjectGeneric.cpp

```

#include "ConfObject.h"
#include "DigitizerErrorObject.h"
#include "LogFile.h"
#include "DigitizerObject.h"
#include "DigitizerObjectGeneric.h"
#include "Analizzatore.h"
#include "DefineGeneral.h"
#include <assert.h>
#include <stdio.h>
#include <CAENDigitizer.h>


DigitizerObjectGeneric::DigitizerObjectGeneric ()
{
  set_board_info = 0;
  logfile = LogFile::Instance ();
}


int
DigitizerObjectGeneric::DigitizerObjectGenericSetAllInformations
 ()
{
    int max_channels = 0;
    int max_groups = 0;

    int FamilyCode = BoardInfo.FamilyCode;

    int FormFactor = BoardInfo.FormFactor;

    int tmp;

    int i = 0;

        switch(FamilyCode) {
        case CAEN_DGTZ_XX724_FAMILY_CODE:
        case CAEN_DGTZ_XX781_FAMILY_CODE:
        case CAEN_DGTZ_XX720_FAMILY_CODE:
        case CAEN_DGTZ_XX721_FAMILY_CODE:
        case CAEN_DGTZ_XX751_FAMILY_CODE:
        case CAEN_DGTZ_XX761_FAMILY_CODE:
        case CAEN_DGTZ_XX731_FAMILY_CODE:
        switch(FormFactor) {
        case CAEN_DGTZ_VME64_FORM_FACTOR:
        case CAEN_DGTZ_VME64X_FORM_FACTOR:
            max_channels = 8;
            break;
        case CAEN_DGTZ_DESKTOP_FORM_FACTOR:
        case CAEN_DGTZ_NIM_FORM_FACTOR:
            max_channels = 4;
            break;
        }
        break;
        case CAEN_DGTZ_XX730_FAMILY_CODE:
        switch(FormFactor) {
        case CAEN_DGTZ_VME64_FORM_FACTOR:
        case CAEN_DGTZ_VME64X_FORM_FACTOR:
            max_channels = 16;
            break;
        case CAEN_DGTZ_DESKTOP_FORM_FACTOR:
```

```
        case CAEN_DGTZ_NIM_FORM_FACTOR:
            max_channels = 8;
            break;
        }
        break;
        case CAEN_DGTZ_XX740_FAMILY_CODE:
        switch(FormFactor) {
        case CAEN_DGTZ_VME64_FORM_FACTOR:
        case CAEN_DGTZ_VME64X_FORM_FACTOR:
            max_channels = 64;
            break;
        case CAEN_DGTZ_DESKTOP_FORM_FACTOR:
        case CAEN_DGTZ_NIM_FORM_FACTOR:
            max_channels = 32;
            break;
        }
        break;
        case CAEN_DGTZ_XX742_FAMILY_CODE:
        switch(FormFactor) {
        case CAEN_DGTZ_VME64_FORM_FACTOR:
        case CAEN_DGTZ_VME64X_FORM_FACTOR:
            //max_channels = 36; ---> sbagliato!!!
            max_groups = 4;
            break;
        case CAEN_DGTZ_DESKTOP_FORM_FACTOR:
        case CAEN_DGTZ_NIM_FORM_FACTOR:
            //max_channels = 16; ---> sbagliato!!!
            max_groups = 2;
            break;
        }
        break;
        default:
        assert("This program cannot be used with this digitizer family\n");
        }

        if (FamilyCode == CAEN_DGTZ_XX740_FAMILY_CODE)
            {

            tmp = max_channels/8;

            if (internal_config.dc_offset != -1)
                {
                for (i=0; i<tmp; i++)
                    {
                    DigitizerObject::DigitizerObjectSetGroupDCOffset (i, internal_config.dc_offset);
                    }
                }
            if (internal_config.trigger_threshold != -1)
                {
                for (i=0; i<tmp; i++)
                    {
                        ret = CAEN_DGTZ_SetGroupTriggerThreshold (handle, i, internal_config.trigger_threshold);
                    logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
                    }
                }

            }
        else
            {

            if (FamilyCode == CAEN_DGTZ_XX742_FAMILY_CODE)
                max_channels = max_groups*8;

            if (internal_config.dc_offset != -1)
                {
                for (i=0; i<max_channels; i++)
                    {
                        DigitizerObject::DigitizerObjectSetChannelDCOffset (i, internal_config.dc_offset);
                    }
                }
            if (internal_config.trigger_threshold != -1)
                {
                for (i=0; i<max_channels; i++)
                    {
                        ret = CAEN_DGTZ_SetChannelTriggerThreshold (handle, i, internal_config.trigger_threshold);
                    logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
                    }
                }
            }

}


```

```
int
DigitizerObjectGeneric::DigitizerObjectGenericSetDecimationFactor
 ()
{
if (BoardInfo.FamilyCode == CAEN_DGTZ_XX740_FAMILY_CODE &&
internal_config.decimation_factor != -1)
    {
  ret = CAEN_DGTZ_SetDecimationFactor (handle, internal_config.
decimation_factor);
  logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
  return 0;
    }
}


int
DigitizerObjectGeneric::DigitizerObjectGenericSetDesMode
 ()
{
    if (((BoardInfo.FamilyCode == CAEN_DGTZ_XX751_FAMILY_CODE) || (
BoardInfo.FamilyCode == CAEN_DGTZ_XX731_FAMILY_CODE)) &&
internal_config.desmod != -1)
    {
    CAEN_DGTZ_EnaDis_t desmodtype;
        if (internal_config.desmod == 1)
        {
        desmodtype = CAEN_DGTZ_ENABLE;
        }
    else
        {
        desmodtype = CAEN_DGTZ_DISABLE;
        }

            ret = CAEN_DGTZ_SetDESMode(handle, desmodtype);
        logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
        return 0;

    }
}


int
DigitizerObjectGeneric::DigitizerObjectGenericSetTestPattern
 ()
{
    if (internal_config.test_pattern == 1)
        {
    ret = CAEN_DGTZ_WriteRegister(handle, CAEN_DGTZ_BROAD_CH_CONFIGBIT_SET_ADD, 1<<3);
    logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
    return 0;
    }
}


int
DigitizerObjectGeneric::DigitizerObjectGenericSetRecordLength
 ()
{
  ret = CAEN_DGTZ_SetRecordLength (handle, internal_config.
record_length);
  logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
  return 0;
}


int
DigitizerObjectGeneric::DigitizerObjectGenericSetMaxNumEventsBLT
 ()
{
  ret =
    CAEN_DGTZ_SetMaxNumEventsBLT (handle, internal_config.
max_num_events_BLT);
  logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
  return 0;
}


int
DigitizerObjectGeneric::DigitizerObjectGenericSetEnableMask
 ()
{
  //Occorre scrivere le informazioni direttamente sui registri perche' le funzioni di libreria non
 funzionano
  if (BoardInfo.FamilyCode == CAEN_DGTZ_XX740_FAMILY_CODE
      || BoardInfo.FamilyCode == CAEN_DGTZ_XX742_FAMILY_CODE)
    {
      if (internal_config.group_enable_mask != -1)
    {
      DigitizerObjectWriteRegister (0x8120,
                    internal_config.group_enable_mask);
    }
    }
  else
    {
      if (internal_config.channel_enable_mask != -1)
    {
      DigitizerObjectWriteRegister (0x8120,
                    internal_config.channel_enable_mask);
    }
    }
}  //int DigitizerObjectGeneric::DigitizerObjectGenericSetEnableMask ()



int
DigitizerObjectGeneric::DigitizerObjectGenericSetExtTriggerInputMode
 ()
{

  switch (internal_config.external_trigger_acquisition_mode
)
    {
    case 0:
      ret =
    CAEN_DGTZ_SetExtTriggerInputMode (handle, CAEN_DGTZ_TRGMODE_ACQ_ONLY);
      logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
      break;

    case 1:
      ret =
    CAEN_DGTZ_SetExtTriggerInputMode (handle,
                      CAEN_DGTZ_TRGMODE_ACQ_AND_EXTOUT);
      logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
      break;

    case 2:
      ret =
    CAEN_DGTZ_SetExtTriggerInputMode (handle, CAEN_DGTZ_TRGMODE_DISABLED);
      logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
      break;
    }

}  //int DigitizerObjectGeneric::DigitizerObjectGenericSetExtTriggerInputMode ()



int
DigitizerObjectGeneric::DigitizerObjectGenericSetDRS4SamplingFrequency
 ()
{
  if (BoardInfo.FamilyCode == CAEN_DGTZ_XX742_FAMILY_CODE)
    {
      switch (internal_config.DSR4_Frequency)
    {
    case 0:
      ret =
        CAEN_DGTZ_SetDRS4SamplingFrequency (handle, CAEN_DGTZ_DRS4_5GHz);
      logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
      break;

    case 1:
      ret =
        CAEN_DGTZ_SetDRS4SamplingFrequency (handle,
                        CAEN_DGTZ_DRS4_2_5GHz);
      logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
      break;

    case 2:
      ret =
        CAEN_DGTZ_SetDRS4SamplingFrequency (handle, CAEN_DGTZ_DRS4_1GHz);
      logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
      break;
    }   //switch(internal_config.DSR4_Frequency)
    }   //if (BoardInfo.FamilyCode == CAEN_DGTZ_XX742_FAMILY_CODE)
}  //int DigitizerObjectGeneric::DigitizerObjectGenericSetDRS4SamplingFrequency ()


int
DigitizerObjectGeneric::DigitizerObjectGenericSetPostTriggerSize
 ()
{
  if (internal_config.post_trigger != -1)
    {
      ret =
    CAEN_DGTZ_SetPostTriggerSize (handle, internal_config.
post_trigger);
      logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
    }
}   //int DigitizerObjectGeneric::DigitizerObjectGenericSetPostTriggerSize ()


int
DigitizerObjectGeneric::DigitizerObjectGenericSetIOLevel
 ()
{

  switch (internal_config.nim_ttl)
    {
    case 0:          //NIM
      ret = CAEN_DGTZ_SetIOLevel (handle, CAEN_DGTZ_IOLevel_NIM);
      logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
      break;

    case 1:          //TTL
      ret = CAEN_DGTZ_SetIOLevel (handle, CAEN_DGTZ_IOLevel_TTL);
      logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
      break;
    }

}   // int DigitizerObjectGeneric::DigitizerObjectGenericSetIOLevel ()



int
DigitizerObjectGeneric::DigitizerObjectGenericSetDCOffset
 ()
{

  int i;

  if (BoardInfo.FamilyCode == CAEN_DGTZ_XX740_FAMILY_CODE)
    {
      for (i = 0; i < MAXGROUPOBJECT; i++)
    {
      if (internal_config.groups[i].set != -1
          && internal_config.groups[i].dc_offset != -1)
        {
          DigitizerObject::DigitizerObjectSetGroupDCOffset
        (internal_config.groups[i].numGroup,
         internal_config.groups[i].dc_offset);
        }
    }   //for (i=0;i<MAXGROUPOBJECT;i++)
    }
  else
    {
      for (i = 0; i < MAXCHANNELOBJECT; i++)
    {
      if (internal_config.channels[i].set != -1
          && internal_config.channels[i].dc_offset != -1)
        {
          DigitizerObject::DigitizerObjectSetChannelDCOffset
        (internal_config.channels[i].numChannel,
         internal_config.channels[i].dc_offset);
        }
    }   //for (i=0;i<MAXCHANNELOBJECT;i++)
    }

}   //int DigitizerObjectGeneric::DigitizerObjectGenericSetDCOffset ()



int
DigitizerObjectGeneric::DigitizerObjectGenericSetSelfTrigger
 ()
{

  if (BoardInfo.FamilyCode == CAEN_DGTZ_XX740_FAMILY_CODE)
    {

      switch (internal_config.self_trigger_enable_mask_mode)
    {
    case 0:
      ret =
        CAEN_DGTZ_SetGroupSelfTrigger (handle, CAEN_DGTZ_TRGMODE_ACQ_ONLY,
                       internal_config.
                       self_trigger_enable_mask);
      logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
      break;

    case 1:
      ret =
        CAEN_DGTZ_SetGroupSelfTrigger (handle,
                       CAEN_DGTZ_TRGMODE_ACQ_AND_EXTOUT,
                       internal_config.
                       self_trigger_enable_mask);
      logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
      break;

    case 2:
      ret =
        CAEN_DGTZ_SetGroupSelfTrigger (handle, CAEN_DGTZ_TRGMODE_DISABLED,
                       internal_config.
                       self_trigger_enable_mask);
      logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
      break;
    }

    }
  else
    {


      switch (internal_config.self_trigger_enable_mask_mode)
    {
    case 0:
      ret =
        CAEN_DGTZ_SetChannelSelfTrigger (handle,
                         CAEN_DGTZ_TRGMODE_ACQ_ONLY,
                         internal_config.
                         self_trigger_enable_mask);
      logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
      break;

    case 1:
      ret =
        CAEN_DGTZ_SetChannelSelfTrigger (handle,
                         CAEN_DGTZ_TRGMODE_ACQ_AND_EXTOUT,
                         internal_config.
                         self_trigger_enable_mask);
      logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
      break;

    case 2:
      ret =
        CAEN_DGTZ_SetChannelSelfTrigger (handle,
                         CAEN_DGTZ_TRGMODE_DISABLED,
                         internal_config.
                         self_trigger_enable_mask);
      logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
      break;
    }

    }

}   //int DigitizerObjectGeneric::DigitizerObjectGenericSetSelfTrigger ()




int
DigitizerObjectGeneric::
DigitizerObjectGenericSetChannelSelfTriggerThreshold ()
{

  int i;

  if (BoardInfo.FamilyCode == CAEN_DGTZ_XX740_FAMILY_CODE)
    {
      for (i = 0; i < MAXGROUPOBJECT; i++)
    {
      if (internal_config.groups[i].set != -1
          && internal_config.groups[i].trigger_threshold != -1)
        {
          ret = CAEN_DGTZ_SetGroupTriggerThreshold
        (handle, internal_config.groups[i].numGroup,
         internal_config.groups[i].trigger_threshold);
          logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
        }
    }   //for (i=0; i<MAXGROUPOBJECT; i++)
    }
  else
    {

      for (i = 0; i < MAXCHANNELOBJECT; i++)
    {
      if (internal_config.channels[i].set != -1
          && internal_config.channels[i].
trigger_threshold != -1)
```

```
          {
            ret = CAEN_DGTZ_SetChannelTriggerThreshold
          (handle, internal_config.channels[i].
numChannel,
           internal_config.channels[i].trigger_threshold);
            logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
          }
      }   //for (i=0;i<MAXCHANNELOBJECT;i++)
      }   //else

}   //int DigitizerObjectGeneric::DigitizerObjectGenericSetChannelSelfTriggerThreshold ()



int
DigitizerObjectGeneric::DigitizerObjectGenericSetFastTriggerDigitizing
  ()
{

  if (BoardInfo.FamilyCode == CAEN_DGTZ_XX742_FAMILY_CODE)
    {

      if (internal_config.enable_fast_trigger_digitizing == 1)
    {
      ret = CAEN_DGTZ_SetFastTriggerDigitizing (handle, CAEN_DGTZ_ENABLE);
      logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
    }

      if (internal_config.enable_fast_trigger_digitizing == 0)
    {
      ret =
        CAEN_DGTZ_SetFastTriggerDigitizing (handle, CAEN_DGTZ_DISABLE);
      logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
    }

      if (internal_config.fast_trigger_acquisition_mode == 0)
    {
      ret =
        CAEN_DGTZ_SetFastTriggerMode (handle, CAEN_DGTZ_TRGMODE_ACQ_ONLY);
      logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
    }

      if (internal_config.fast_trigger_acquisition_mode == 2)
    {
      ret =
        CAEN_DGTZ_SetFastTriggerMode (handle, CAEN_DGTZ_TRGMODE_DISABLED);
      logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
    }

      if (internal_config.fasts[0].set != -1
      && internal_config.fasts[0].dc_offset != -1)
    {
      ret =
        CAEN_DGTZ_SetGroupFastTriggerDCOffset (handle,
                           internal_config.fasts[0].
                           numFast,
                           internal_config.fasts[0].
                           dc_offset);
      logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
    }

      if (internal_config.fasts[1].set != -1
      && internal_config.fasts[1].dc_offset != -1)
    {
      ret =
        CAEN_DGTZ_SetGroupFastTriggerDCOffset (handle,
                           internal_config.fasts[1].
                           numFast,
                           internal_config.fasts[1].
                           dc_offset);
      logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
    }

      if (internal_config.fasts[0].set != -1
      && internal_config.fasts[0].trigger_threshold != -1)
    {
      ret =
        CAEN_DGTZ_SetGroupFastTriggerThreshold (handle,
                            internal_config.fasts[0].
                            numFast,
                            internal_config.fasts[0].
                            trigger_threshold);
      logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
    }

      if (internal_config.fasts[1].set != -1
      && internal_config.fasts[1].trigger_threshold != -1)
    {
      ret = CAEN_DGTZ_SetGroupFastTriggerThreshold
        (handle, internal_config.fasts[1].numFast,
         internal_config.fasts[1].trigger_threshold);
      logfile->LogFileWrite (ret, __FILE__, __func__, __LINE__);
    }

    }   //if (internal_config.BoardInfo.FamilyCode == CAEN_DGTZ_XX742_FAMILY_CODE){

}   //int DigitizerObjectGeneric::DigitizerObjectGenericSetFastTriggerDigitizing ()
```

6.43 DigitizerObjectGeneric.h File Reference

```
#include <assert.h>
#include <stdio.h>
#include <CAENDigitizer.h>
```

Classes

- class DigitizerObjectGeneric

  The DigitizerObjectGeneric class extends the DigitizerObject class whith methods for setting the digitizer using "internal_config" attribute.

6.44 DigitizerObjectGeneric.h

```

#include <assert.h>
#include <stdio.h>
#include <CAENDigitizer.h>

class DigitizerObjectGeneric:public DigitizerObject
{
public:

  int DigitizerObjectGenericSetRecordLength ();

  int DigitizerObjectGenericSetMaxNumEventsBLT ();

  int DigitizerObjectGenericSetEnableMask ();

  int DigitizerObjectGenericSetFastTriggerDigitizing ();

  int DigitizerObjectGenericSetSelfTrigger ();

  int DigitizerObjectGenericSetChannelSelfTriggerThreshold
 ();


  int DigitizerObjectGenericSetDCOffset ();

  int DigitizerObjectGenericWriteRegister ();

  int DigitizerObjectGenericReadRegister ();

  int DigitizerObjectGenericGetDRS4SamplingFrequency ();

  int DigitizerObjectGenericSetDRS4SamplingFrequency ();

  int DigitizerObjectGenericSetChannelDCOffset ();

  int DigitizerObjectGenericSetGroupDCOffset ();

  int DigitizerObjectGenericSetTriggerPolarity ();

  int DigitizerObjectGenericSetPostTriggerSize ();

  int DigitizerObjectGenericSetIOLevel ();

  int DigitizerObjectGenericSetAcquisitionMode ();

  int DigitizerObjectGenericSetExtTriggerInputMode ();

  int DigitizerObjectGenericSetSWTriggerMode ();

  int DigitizerObjectGenericSetDecimationFactor ();

  int DigitizerObjectGenericSetDesMode ();

  int DigitizerObjectGenericSetTestPattern ();

  int DigitizerObjectGenericSetAllInformations ();

  DigitizerObjectGeneric ();
};
```

6.45 DigitizerStateMachine.cpp File Reference

```
#include "DefineGeneral.h"
#include "X742DecodeRoutines.h"
#include "ConfObject.h"
#include "ApplicationSetup.h"
#include "DigitizerErrorObject.h"
#include "LogFile.h"
#include "DigitizerObject.h"
#include "Analizzatore.h"
#include "DigitizerObjectGeneric.h"
#include "RawData.h"
#include "DigitizerStateMachine.h"
#include "OutputModule.h"
#include <time.h>
#include <pthread.h>
#include <assert.h>
#include <stdio.h>
#include <CAENDigitizer.h>
#include <thread>
#include <mutex>
#include <condition_variable>
```

6.46 DigitizerStateMachine.cpp

```

#include "DefineGeneral.h"
#include "X742DecodeRoutines.h"
#include "ConfObject.h"
#include "ApplicationSetup.h"
#include "DigitizerErrorObject.h"
#include "LogFile.h"
#include "DigitizerObject.h"
#include "Analizzatore.h"
#include "DigitizerObjectGeneric.h"
#include "RawData.h"
#include "DigitizerStateMachine.h"
#include "OutputModule.h"
#include <time.h>
#include <pthread.h>
#include <assert.h>
#include <stdio.h>
#include <CAENDigitizer.h>
#include <thread>
#include <mutex>
#include <condition_variable>

using namespace std;

void
DigitizerStateMachine::Produttore ()
{
//Codice per testare il tempo.
/*

```

```
  struct timespec start, stop;
  unsigned long accum;
  unsigned long parz_clock;
  unsigned long parz=1000000000L;

  double accum2;
*/
  unique_lock<mutex> ReservedConsumerDispatcherInputAreaHandle(ReservedConsumerDispatcherInputArea,
defer_lock);
  unique_lock<mutex> ReservedPreprocessingInputAreaHandle(ReservedPreprocessingInputArea, defer_lock);
  unique_lock<mutex> ReservedVisualizationInputAreaHandle(ReservedVisualizationInputArea, defer_lock);

  int i;
  RawData tmp;
  tmp.RawDataSet (digitizer);

  //Codice per testare il tempo.
  //clock_gettime(CLOCK_REALTIME, &start);

  while (go_general)
    {
  //Codice per testare il tempo.
  /*clock_gettime(CLOCK_REALTIME, &stop);

          parz_clock = ((unsigned long)stop.tv_sec - (unsigned long)start.tv_sec) - 1 ;
          if (parz_clock>=0) {accum=parz_clock*1000000000L+((parz-start.tv_nsec)+stop.tv_nsec);}
          else accum = stop.tv_sec - start.tv_sec;
          //printf( "%lf\n", accum );
                accum2=(double)accum;
        accum2=accum2/1000000L;
            if(tmp.bsize>0)
        fprintf(stderr, "\t%.6lf\n", accum2);

      clock_gettime(CLOCK_REALTIME, &start);
  */

      tmp.RawDataRead ();

      if (tmp.bsize > 0)
    {

      ReservedVisualizationInputAreaHandle.lock();
      //fprintf(stderr, "num_mex_raw_data:%d\n", num_mex_raw_data);
      if (go_raw_data && go_general)
        {
          if (num_mex_raw_data < RAWDATAQUEUE)
        {
          circular_buffer_raw_data[coda_raw_data] = tmp;
          if (coda_raw_data == RAWDATAQUEUE - 1)
            coda_raw_data = 0;
          else
            coda_raw_data = coda_raw_data + 1;
          num_mex_raw_data++;
          BlockedConsumerInput.notify_one();
        }
        }   // go_raw_data
      ReservedVisualizationInputAreaHandle.unlock();



    ReservedPreprocessingInputAreaHandle.lock();
    //fprintf(stderr, "num_mex_preprocessing:%d\n", num_mex_preprocessing);
      if (go_preprocessing && go_general)
        {
          if (num_mex_preprocessing < PREPROCESSINGQUEUE)
        {
          circular_buffer_preprocessing[coda_preprocessing] = tmp;

          if (coda_preprocessing == PREPROCESSINGQUEUE - 1)
            coda_preprocessing = 0;
          else
            coda_preprocessing = coda_preprocessing + 1;
          num_mex_preprocessing++;
          Acquisition_Cond1.notify_one();
        }
        }   // go_preprocessing
    ReservedPreprocessingInputAreaHandle.unlock();



          ReservedVisualizationInputAreaHandle.lock();
      //fprintf(stderr, "num_mex_visualization:%d\n", num_mex_visualization);
      if (go_visualization && go_general)
        {
          if (num_mex_visualization < VISUALIZATIONQUEUE)
        {
          circular_buffer_visualization[coda_visualization] = tmp;

          if (coda_visualization == VISUALIZATIONQUEUE - 1)
            coda_visualization = 0;
          else
            coda_visualization = coda_visualization + 1;
          num_mex_visualization++;
          Acquisition_Cond2.notify_one();
        }
        }   //go_visualization
          ReservedVisualizationInputAreaHandle.unlock();
    }   //if (tmp.bsize > 0)
    }   // go_general

//segnalo la fine dell’acquisizione a tutti i threads
  for (i = 0; i < 10; i++)
    {
      Acquisition_Cond2.notify_one();
      Acquisition_Cond1.notify_one();
      BlockedConsumerInput.notify_one();
      raw_cond.notify_one();
      pre_cond.notify_one();
      vis_cond.notify_one();
    }
  //fprintf (stderr, "Esco dal Reader Thread\n");

}

void
DigitizerStateMachine::Consumatore_Dispatcher ()
{
  ApplicationSetup *application_setup;
  application_setup = ApplicationSetup::Instance ();

  RawData tmp;
  tmp.RawDataSet (digitizer);

  unique_lock<mutex> ReservedConsumerDispatcherInputAreaHandle(ReservedConsumerDispatcherInputArea,
std::defer_lock);

    while (go_general)
        {

        if (go_raw_data && go_general)
            {


            ReservedConsumerDispatcherInputAreaHandle.lock();

            if (num_mex_raw_data == 0)
                {
                    BlockedConsumerInput.wait(ReservedConsumerDispatcherInputAreaHandle);
                }

            if (go_general != 0 && go_raw_data != 0)
                    {

            tmp = circular_buffer_raw_data[testa_raw_data];

            ReservedConsumerDispatcherInputAreaHandle.unlock();


            //fprintf(stderr, "scrivo\n");
            if (tmp.bsize > 0)
                    tmp.RawDataWriteOnFile (application_setup->
                ApplicationSetupGetDataFilePunt (),
                application_setup->
                ApplicationSetupGetDataFileSizePunt ());


            ReservedConsumerDispatcherInputAreaHandle.lock();

            if (testa_raw_data == RAWDATAQUEUE - 1)
                    testa_raw_data = 0;
            else
                    testa_raw_data = testa_raw_data + 1;

            num_mex_raw_data--;
                }
            ReservedConsumerDispatcherInputAreaHandle.unlock();


            }
        else
            {
            ReservedConsumerDispatcherInputAreaHandle.lock();
            raw_cond.wait(ReservedConsumerDispatcherInputAreaHandle);
            ReservedConsumerDispatcherInputAreaHandle.unlock();
            }
        }  //go_general
//fprintf (stderr, "Esco dal Thread RawData\n");
}


void
DigitizerStateMachine::Preprocessing ()
{

  int i, j;
  uint32_t numEvents;
  int conta = 0;
  char *evtptr;
  CAEN_DGTZ_EventInfo_t eventInfo;
  RawData tmp;
  tmp.RawDataSet (digitizer);

  unique_lock<mutex> ReservedPreprocessingInputAreaHandle(ReservedPreprocessingInputArea, defer_lock);

    while (go_general)
        {

        if (go_preprocessing && go_general)
            {


            ReservedPreprocessingInputAreaHandle.lock();

            if (num_mex_preprocessing == 0)
                {
                Acquisition_Cond1.wait(ReservedPreprocessingInputAreaHandle);
                }

            if (go_general != 0 && go_preprocessing != 0)
                {

            tmp = circular_buffer_preprocessing[testa_preprocessing];

            ReservedPreprocessingInputAreaHandle.unlock();


            //fprintf(stderr, "Decodifico\n");
            tmp.RawDataDecode ();


            ReservedPreprocessingInputAreaHandle.lock();

            if (testa_preprocessing == PREPROCESSINGQUEUE - 1)
                testa_preprocessing = 0;
            else
                testa_preprocessing = testa_preprocessing + 1;

            num_mex_preprocessing--; }

            ReservedPreprocessingInputAreaHandle.unlock();

            }
        else
            {
            ReservedPreprocessingInputAreaHandle.lock();
            pre_cond.wait(ReservedPreprocessingInputAreaHandle);
            ReservedPreprocessingInputAreaHandle.unlock();
            }
        }   // go_general
}


void
DigitizerStateMachine::Visualization ()
{
  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  RawData tmp;
  tmp.RawDataSet (digitizer);

  FILE *gnuplot;

  char stringa[STANDARDBUFFERLIMIT];
  bzero (stringa, STANDARDBUFFERLIMIT);
  snprintf(stringa, STANDARDBUFFERLIMIT, "%s -persist 2>/dev/null", digitizer.
internal_config.gnuplot);

  gnuplot = NULL;
  gnuplot = popen (stringa, "w");

  unique_lock<mutex> ReservedVisualizationInputAreaHandle(ReservedVisualizationInputArea, defer_lock);

//Il codice seguente serve per gestire il caso in cui l'utente non abbia installato gnuplot o abbia
 specificato male il comando
//per lanciarlo nel configuration file. Occorre gestire questa situazione perche' si rischia, altrimenti,
 l'arresto del programma senza nessun
//messaggio di errore rendendo il debugging difficile.
  int gnuplot_error_code;
  gnuplot_error_code = pclose(gnuplot);

  if (gnuplot_error_code != 0)
    {
    output_module->Output("Error, you have not specify gnuplot command correctly in the configuration
 file\n");
    }

  gnuplot = NULL;
  gnuplot = popen (stringa, "w");

    while (go_general)
        {

        if (go_visualization && go_general)
            {

            ReservedVisualizationInputAreaHandle.lock();

            if (num_mex_visualization == 0)
                Acquisition_Cond2.wait(ReservedVisualizationInputAreaHandle);
            if (go_general != 0 && go_visualization != 0)
                {
                tmp = circular_buffer_visualization[testa_visualization];

            ReservedVisualizationInputAreaHandle.unlock();


            //fprintf(stderr, "plotto\n");
            if (tmp.bsize > 0 && gnuplot_error_code == 0)
                tmp.RawDataPlot (".plot_data.txt", gnuplot);


            ReservedVisualizationInputAreaHandle.lock();

            if (testa_visualization == VISUALIZATIONQUEUE - 1)
                testa_visualization = 0;
            else
                    testa_visualization = testa_visualization + 1;

            num_mex_visualization--; }

            ReservedVisualizationInputAreaHandle.unlock();
            }
        else
            {
            ReservedVisualizationInputAreaHandle.lock();
            vis_cond.wait(ReservedVisualizationInputAreaHandle);
            ReservedVisualizationInputAreaHandle.unlock();
            }

        }  // go_general

  pclose (gnuplot);
}

DigitizerStateMachine::DigitizerStateMachine ()
{
  imset = 0;
  imstarted = 0;
  go_general = 0;
  go_raw_data = 0;
  go_preprocessing = 0;
  go_visualization = 0;
}

void
DigitizerStateMachine::DigitizerStateMachineInit (const
char *conf_file)
{
  digitizer.DigitizerObjectSetConfigStructureInit (conf_file);
  //Se la variabile imset e' impostata a 0 e viene eseguito il metodo
 DigitizerStateMachine::DigitizerStateMachineStartReading,
  //viene stampato un messaggio di errore.
  imset = 1;

  digitizer.DigitizerObjectOpen ();
  digitizer.DigitizerObjectReset ();
  digitizer.DigitizerObjectGetInfo ();
  digitizer.PrintBoardInfo ();
}

void
DigitizerStateMachine::DigitizerStateMachineSetup (const
char *conf_file)
{
  int data;
  digitizer.DigitizerObjectReset ();
  digitizer.DigitizerObjectSetConfigStructureSetup (conf_file);
  digitizer.DigitizerObjectGenericSetRecordLength ();
  digitizer.DigitizerObjectGenericSetMaxNumEventsBLT ();
  digitizer.DigitizerObjectSetAcquisitionMode (CAEN_DGTZ_SW_CONTROLLED);
  digitizer.DigitizerObjectGenericSetExtTriggerInputMode ();
  digitizer.DigitizerObjectGenericSetEnableMask ();
  digitizer.DigitizerObjectGenericSetDRS4SamplingFrequency ();
  digitizer.DigitizerObjectGenericSetPostTriggerSize ();
  digitizer.DigitizerObjectGenericSetChannelSelfTriggerThreshold ();
  digitizer.DigitizerObjectGenericSetIOLevel ();
  digitizer.DigitizerObjectGenericSetDCOffset ();
  digitizer.DigitizerObjectGenericSetSelfTrigger ();
  digitizer.DigitizerObjectGenericSetFastTriggerDigitizing ();
  digitizer.DigitizerObjectGenericSetDecimationFactor ();
  digitizer.DigitizerObjectGenericSetDesMode ();
  digitizer.DigitizerObjectGenericSetTestPattern ();
  digitizer.DigitizerObjectGenericSetAllInformations ();
}

void
DigitizerStateMachine::DigitizerStateMachineRawDataInit
 ()
{
  int i = 0;
  num_mex_raw_data = 0;
  coda_raw_data = 0;
  testa_raw_data = 0;

  num_mex_preprocessing = 0;
  coda_preprocessing = 0;
  testa_preprocessing = 0;

  num_mex_visualization = 0;
  coda_visualization = 0;
  testa_visualization = 0;

  for (i = 0; i < RAWDATAQUEUE; i++)
    {
      circular_buffer_raw_data[i].RawDataSet (digitizer);
    }

  for (i = 0; i < PREPROCESSINGQUEUE; i++)
    {
      circular_buffer_preprocessing[i].RawDataSet (digitizer);
    }

  for (i = 0; i < VISUALIZATIONQUEUE; i++)
    {
      circular_buffer_visualization[i].RawDataSet (digitizer);
    }

  go_general = 1;

  visualization_thread = new thread (&DigitizerStateMachine::Visualization
, this);

  preprocessing_thread = new thread (&DigitizerStateMachine::Preprocessing
, this);

  consumatore_thread = new thread (&
DigitizerStateMachine::Consumatore_Dispatcher, this);

  produttore_thread = new thread (&DigitizerStateMachine::Produttore, this
);

}


void
DigitizerStateMachine::DigitizerStateMachineStartReading
 ()
{

  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  if (imset == 0)
    {
      output_module->Output("DigitizerStateMachine: Error, you have not set the object, use
 DigitizerStateMachineInit(const char *conf_file) method.\n");
    }

  if (imstarted == 0)
    {
      imstarted = 1;
      digitizer.DigitizerObjectSWStartAcquisition ();
      DigitizerStateMachine::DigitizerStateMachineRawDataInit
 ();
      output_module->Output("DigitizerStateMachine: Inizio la lettura dei dati\n");
    }
  else
    {
      output_module->Output("DigitizerStateMachine: Error, you have already insert start command \n")
;
    }

}


void
DigitizerStateMachine::DigitizerStateMachineQuit ()
{

  OutputModule *output_module;
  output_module = OutputModule::Instance ();
  output_module->Output("DigitizerStateMachine: Esco dalla sessione di acquisizione\n");
  DigitizerStateMachine::DigitizerStateMachineStopReading
 ();
  digitizer.DigitizerObjectClose ();

}

void
DigitizerStateMachine::DigitizerStateMachineStopReading
 ()
{

  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  output_module->Output("DigitizerStateMachine: Interrompo la lettura dei dati\n");

  imstarted = 0;
  go_general = 0;


  //DigitizerStateMachine::DigitizerStateStopRawDataWriting();
  BlockedConsumerInput.notify_one();


  //DigitizerStateMachine::DigitizerStateStopPreprocessing();
  Acquisition_Cond1.notify_one();


  //DigitizerStateMachine::DigitizerStateStopVisualization();
  Acquisition_Cond2.notify_one();

  raw_cond.notify_one();
  vis_cond.notify_one();
  pre_cond.notify_one();

  digitizer.DigitizerObjectSWStopAcquisition ();

}

void
DigitizerStateMachine::DigitizerStateMachineSendSWTrigger
 ()
{
  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  output_module->Output("DigitizerStateMachine: invio il software trigger\n");

  CAEN_DGTZ_SendSWtrigger (digitizer.handle);
}

void
DigitizerStateMachine::DigitizerStateStartPreprocessing
 ()
{
  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  output_module->Output("DigitizerStateMachine: Starting Preprocessing thread\n");

  go_preprocessing = 1;
  pre_cond.notify_one();
}

void
DigitizerStateMachine::DigitizerStateStartVisualization
  ()
{
  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  output_module->Output("Starting Visualization thread\n");

  go_visualization = 1;
  vis_cond.notify_one();
}

void
DigitizerStateMachine::DigitizerStateStopPreprocessing
  ()
{
  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  output_module->Output("DigitizerStateMachine: Stopping Preprocessing thread\n");

  go_preprocessing = 0;
}

void
DigitizerStateMachine::DigitizerStateStartRawDataWriting
  ()
{
  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  output_module->Output("DigitizerStateMachine: Starting RawData thread\n");

  go_raw_data = 1;
  raw_cond.notify_one();
}

void
DigitizerStateMachine::DigitizerStateStopRawDataWriting
  ()
{
  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  output_module->Output("DigitizerStateMachine: Stopping RawData thread\n");

  go_raw_data = 0;
}

void
DigitizerStateMachine::DigitizerStateStopVisualization
  ()
{
  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  output_module->Output("DigitizerStateMachine: Stopping Visualization thread\n");

  go_visualization = 0;
}

void
DigitizerStateMachine::DigitizerStateMachinePrintStatus
  ()
{
  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  if (go_general)
    {
      output_module->Output("Data acquisition thread ON\n");
    }
  else
    {
      output_module->Output("Data acquisition thread OFF\n");
    }

  if (go_preprocessing)
    {
      output_module->Output("Preprocessing thread ON\n");
    }
  else
    {
      output_module->Output("Preprocessing thread OFF\n");
    }

  if (go_raw_data)
    {
      output_module->Output("Raw data thread ON\n");
    }
  else
    {
      output_module->Output("Raw data thread OFF\n");
    }

  if (go_visualization)
    {
      output_module->Output("Visualization thread ON\n");
    }
  else
    {
      output_module->Output("Visualization thread OFF\n");
    }

}
```

6.47 DigitizerStateMachine.h File Reference

```
#include "DefineGeneral.h"
#include <assert.h>
#include <stdio.h>
#include <pthread.h>
#include <CAENDigitizer.h>
#include <thread>
#include <mutex>
#include <condition_variable>
```

Classes

- class DigitizerStateMachine

6.48 DigitizerStateMachine.h

```
#include "DefineGeneral.h"
#include <assert.h>
#include <stdio.h>
#include <pthread.h>
#include <CAENDigitizer.h>
#include <thread>
#include <mutex>
#include <condition_variable>

using namespace std;

class DigitizerStateMachine
{

public:

  int num_mex_raw_data;

  int num_mex_preprocessing;

  int num_mex_visualization;

  int testa_raw_data;

  int testa_preprocessing;

  int testa_visualization;

  int coda_raw_data;

  int coda_preprocessing;

  int coda_visualization;

  int go_general;

  int go_preprocessing;

  int go_raw_data;

  int go_visualization;

  DigitizerObjectGeneric digitizer;

  RawData circular_buffer_raw_data[RAWDATAQUEUE];

  RawData circular_buffer_preprocessing[PREPROCESSINGQUEUE];

  RawData circular_buffer_visualization[VISUALIZATIONQUEUE];

  void DigitizerStateMachineSetup (const char *conf_file);

  void DigitizerStateMachineStartReading ();

  void DigitizerStateMachinePrintStatus ();

  void DigitizerStateMachineStopReading ();

  void DigitizerStateMachineRawDataInit ();

  void DigitizerStateMachineQuit ();

  void DigitizerStateMachineSendSWTrigger ();

  void DigitizerStateStartPreprocessing ();

  void DigitizerStateStartVisualization ();

  void DigitizerStateStopPreprocessing ();

  void DigitizerStateStartRawDataWriting ();

  void DigitizerStateStopRawDataWriting ();

  void DigitizerStateStopVisualization ();

  void Produttore ();

  void Consumatore_Dispatcher ();

  void Preprocessing ();

  void Visualization ();

  thread * produttore_thread;

  thread * consumatore_thread;

  thread * preprocessing_thread;

  thread * visualization_thread;

  mutex ReservedConsumerDispatcherInputArea;

  condition_variable BlockedProducerInput;

  condition_variable BlockedConsumerInput;

  mutex ReservedPreprocessingInputArea;

  mutex ReservedVisualizationInputArea;

  condition_variable Acquisition_Cond1;

  condition_variable Acquisition_Cond2;

  condition_variable raw_cond;

  condition_variable pre_cond;

  condition_variable vis_cond;

  void DigitizerStateMachineInit (const char *conf_file);

```

```
  int imstarted;

  int imset;

    DigitizerStateMachine ();
};
```

6.49 Input.cpp File Reference

```
#include "DefineGeneral.h"
#include "DefineCommands.h"
#include "TcpUser.h"
#include "ConfObject.h"
#include "DigitizerErrorObject.h"
#include "LogFile.h"
#include "DigitizerObject.h"
#include "DigitizerObjectGeneric.h"
#include "RawData.h"
#include "DigitizerStateMachine.h"
#include "ApplicationSetup.h"
#include "Input.h"
#include "CommunicationObject.h"
#include "DigitizerFlowControl.h"
#include "ConfigurationConsistence.h"
#include "AnalizzatoreUtils.h"
#include "Analizzatore.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <thread>
#include <mutex>
#include <pthread.h>
#include <condition_variable>
```

6.50 Input.cpp

```

#include "DefineGeneral.h"
#include "DefineCommands.h"
#include "TcpUser.h"
#include "ConfObject.h"
#include "DigitizerErrorObject.h"
#include "LogFile.h"
#include "DigitizerObject.h"
#include "DigitizerObjectGeneric.h"
#include "RawData.h"
#include "DigitizerStateMachine.h"
#include "ApplicationSetup.h"
#include "Input.h"
#include "CommunicationObject.h"
#include "DigitizerFlowControl.h"
#include "ConfigurationConsistence.h"
#include "AnalizzatoreUtils.h"
#include "Analizzatore.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <thread>
#include <mutex>
#include <pthread.h>
#include <condition_variable>

using namespace std;

Input::Input ()
{
  int i;
  num_mex = 0;
  go = 1;
  input_buffer = (char *) malloc (STANDARDBUFFERLIMIT);
  for (i = 0; i < STANDARDBUFFERLIMIT; i++)
    input_buffer[i] = '\0';
  producer_thread = new thread (&Input::Producer, this);
}

void
Input::Producer ()
{
  int i;
  char input_buffer_private[STANDARDBUFFERLIMIT];

  //go is set to 1 by the costructor. The finish() method set it to 0.
  while (go)
    {
      for (i = 0; i < STANDARDBUFFERLIMIT; i++)
    {
      input_buffer_private[i] = '\0';
    }

      fgets (input_buffer_private, 99, stdin);  //getting the input from stdin

      for (i = 0; i < STANDARDBUFFERLIMIT; i++)
    {
      if (input_buffer_private[i] == '\n')
        input_buffer_private[i] = '\0';
    }

      unique_lock<mutex> lk(mutex1);    //assure data consistency

      for (i = 0; i < STANDARDBUFFERLIMIT; i++)
    {
      input_buffer[i] = input_buffer_private[i];
    }

      pthread_mutex_lock (&DigitizerFlowControl::input_flow_mutex);
      pthread_cond_signal (&DigitizerFlowControl::input_flow_cond);
      pthread_mutex_unlock (&DigitizerFlowControl::input_flow_mutex);

      num_mex = 1;  // a message is ready

      lk.unlock();
    }


}


void
Input::GetInput (char *input_buffer_sending)
{
  int i;
      unique_lock<mutex> lk(mutex1);    //assure data consistency

  if (num_mex == 0) // if no message set the input_buffer_sending to zero
    {
      for (i = 0; i < STANDARDBUFFERLIMIT; i++)
    input_buffer_sending[i] = '\0';

    }

  else      // if there is a message, copy it into input_buffer_sending
    {
      for (i = 0; i < STANDARDBUFFERLIMIT; i++)
    {
      input_buffer_sending[i] = input_buffer[i];
    }
      num_mex = 0;  //no message ready
    }
}

void
Input::Finish ()
{
  go = 0; // stop the producer thread
}
```

6.51 Input.h File Reference

```
#include <thread>
```

```
#include <mutex>
#include <condition_variable>
```

Classes

- class Input

6.52 Input.h

```

#include <thread>
#include <mutex>
#include <condition_variable>

using namespace std;

class Input
{
public:

  char *input_buffer;

  void GetInput (char *input_buffer_sending);

  void Finish ();

  thread * producer_thread;

  mutex mutex1;

  int go;

  int num_mex;

  void Producer ();

  Input ();
};
```

6.53 LogFile.cpp File Reference

```
#include "ConfObject.h"
#include "ApplicationSetup.h"
#include "DigitizerErrorObject.h"
#include "LogFile.h"
#include "DefineGeneral.h"
#include "OutputModule.h"
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

6.54 LogFile.cpp

```

#include "ConfObject.h"
#include "ApplicationSetup.h"
#include "DigitizerErrorObject.h"
#include "LogFile.h"
#include "DefineGeneral.h"
```

```
#include "OutputModule.h"
#include<sys/socket.h>
#include<arpa/inet.h>
#include<unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


LogFile *
  LogFile::log_file_pInstance = NULL;


LogFile *
LogFile::Instance ()
{
  if (!log_file_pInstance)  // Only allow one instance of class to be generated.
    log_file_pInstance = new LogFile ();

  return log_file_pInstance;
}


void
LogFile::LogFileMessageOn ()
{

flag = 1;

}


void
LogFile::LogFileMessageOff ()
{

flag = 1;

}

void
LogFile::LogFileSet (const char *log_file_arg)
{

  flag = 0;

  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  log_file_path = (const char *) malloc (strlen (log_file_arg) + 1);
  strcpy ((char *) log_file_path, (char *) log_file_arg);
  log_file_punt = fopen (log_file_path, "w");

  if (log_file_punt == NULL)
    {
      output_module->Output("Warning, I can't access log file.\n");
      can_write = 0;

    }
  else
    {
      output_module->Output("Accessing log file...\n");
      can_write = 1;
    }
}                //void LogFile::LogFileSet(const char * log_file_arg)


void
LogFile::LogFileWriteString (const char *string)
{
  if (can_write)
    {
      fprintf (log_file_punt, string);
    }
}


void
LogFile::LogFileWrite (CAEN_DGTZ_ErrorCode ret_arg, const char *file,
                const char *func, int line)
{
  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  if (can_write)
    ret_debug =
      my_error.DigitizerErrorObjectDebuggingLog (ret_arg, file, func, line,
                            log_file_punt);

  if (ret_debug && flag == 1)
    {
    output_module->Output("Some error occurred, digit \"more\" for informations\n");
        flag = 0;
    }

} //void LogFile::LogFileWrite(CAEN_DGTZ_ErrorCode ret_arg, const char *file, const char *func, int line)


void
LogFile::LogFileRead ()
{
  OutputModule *output_module;
  output_module = OutputModule::Instance ();

  char buffer[999];
  bzero (buffer, 999);
  char ch;

  int i = 0;
  int flag = 0;

  FILE *read_log;
  read_log = fopen (log_file_path, "r");


  if (read_log != NULL)
    {
      fseek (read_log, 0, SEEK_SET);
      while ((ch = fgetc (read_log)) != EOF)
    {
      if (i < 997)
        {
          buffer[i] = ch;
          i++;
          flag = 1;
        }
      else
        {
          buffer[i] = ch;
          buffer[998] = '\0';
          output_module->Output((const char *) buffer);
          bzero (buffer, 999);
          flag = 0;
          i = 0;
        }           // } else {
    }           //while( ( ch = fgetc(read_log) ) != EOF )

      if (flag == 1)
    {
      buffer[i] = '\0';
      output_module->Output((const char *) buffer);
      bzero (buffer, 999);
    }
      fclose (read_log);    // N.B. la fclose deve stare dentro senno' capita una segmentation fault!!!
    }   // if (read_log != NULL)

}   //void LogFile::LogFileRead()
```

6.55 LogFile.h File Reference

```
#include <stdio.h>
```

Classes

- class LogFile

  The LogFile singleton is used to write the error codes retrieved by the CAENDigitizer function to the logfile indicates by the log_file_path attribute.

6.56 LogFile.h

```
#include <stdio.h>

class LogFile
{
private:
  int ret_debug;

  int can_write;

  int flag;

  FILE *log_file_punt;

  static LogFile *log_file_pInstance;

  DigitizerErrorObject my_error;

public:
  const char *log_file_path;

  static LogFile *Instance ();

  void LogFileSet (const char *log_file_arg);

  void LogFileWrite (CAEN_DGTZ_ErrorCode ret_arg, const char *file,
              const char *func, int line);

  void LogFileWriteString (const char *string);

  void LogFileRead ();

  void LogFileMessageOn ();

  void LogFileMessageOff ();
};
```

6.57 Main.c File Reference

This file contains the program main.

```
#include "X742DecodeRoutines.h"
#include "TcpUser.h"
#include "ConfObject.h"
#include "DigitizerErrorObject.h"
#include "LogFile.h"
#include "DigitizerObject.h"
#include "DigitizerObjectGeneric.h"
#include "RawData.h"
#include "DigitizerStateMachine.h"
#include "Analizzatore.h"
#include "ApplicationSetup.h"
#include "Input.h"
#include "CommunicationObject.h"
#include "DigitizerFlowControl.h"
#include <CAENDigitizer.h>
#include <stdio.h>
#include <unistd.h>
#include <ctype.h>
```

Macros

- #define ONLY_TCP 1

Functions

- int main (int argc, char ∗∗argv)

  The main function of the application.

6.57.1 Detailed Description

This file contains the program main.

Definition in file Main.c.

6.57.2 Macro Definition Documentation

6.57.2.1 #define ONLY_TCP 1

Definition at line 37 of file Main.c.

6.57.3 Function Documentation

6.57.3.1 int main (int *argc,* char ∗∗ *argv*)

The main function of the application.

Parameters

argc	is the number of the main arguments
argv	are the main arguments

Returns

int

Definition at line 46 of file Main.c.

6.58 Main.c

```
00001
00018 #include "X742DecodeRoutines.h"
00019 #include "TcpUser.h"
00020 #include "ConfObject.h"
00021 #include "DigitizerErrorObject.h"
00022 #include "LogFile.h"
00023 #include "DigitizerObject.h"
00024 #include "DigitizerObjectGeneric.h"
00025 #include "RawData.h"
00026 #include "DigitizerStateMachine.h"
00027 #include "Analizzatore.h"
00028 #include "ApplicationSetup.h"
00029 #include "Input.h"
00030 #include "CommunicationObject.h"
00031 #include "DigitizerFlowControl.h"
00032 #include <CAENDigitizer.h>
00033 #include <stdio.h>
00034 #include <unistd.h>
00035 #include <ctype.h>
00036
00037 #define ONLY_TCP 1
00038
00045 int
00046 main (int argc, char **argv)
00047 {
00048
00049   //Se l'utente sceglie di avviare il programma in modalita' ONLY_TCP (flag -m tcp), occorre eseguire il
       programma in modalita' demone.
```

```
 //Di default il valore della variabile e' zero cosi' l'espressione process_id == 0 e' verificata anche senza assegnare a process_id il risultato
 //della system call fork() che viene chiamata solo se il programma viene eseguito con il flag -m tcp (cioe' l'unica modalita' di accettazione dell'input e di
 //produzione dell'output e' quella tramite tcp).
 pid_t process_id = 0;

 //L'oggetto di tipo ApplicationSetup serve per salvare le impostazioni che l'utente ha inserito tramite flags nella stringa di esecuzione del programma.
 //ApplicationSetup e' un singleton, quindi e' necessario ottenerne un riferimento con il metodo Instance().
 ApplicationSetup *main_application_setup;
 main_application_setup = ApplicationSetup::Instance ();

 //Il metodo ApplicationSetupSet copia nello scope di ApplicationSetup gli argomenti del main argc e argv, li analizza e salva le informazioni ottenute.
 //Gli altri oggetti del programma potranno accedere alle informazioni ottenute ottenendo un'istanza di ApplicationSetup.
 main_application_setup->ApplicationSetupSet (argc, argv);

 //Se l'utente ha avviato il programma con il flag -m tcp, esso deve essere eseguito in modalita' demone.
 if (main_application_setup->input_mode == ONLY_TCP)
   {
     process_id = fork ();
   }

 //L'espressione risulta vera se e' letta dal processo figlio o da processo padre eseguito senza il flag -m tcp
 if (process_id == 0)
   {
     //Creo e avvio l'oggetto digitizer_flow_control_main di tipo DigitizerFlowControl: l'oggetto in questione gestisce l'intero flusso di esecuzione del programma.
     DigitizerFlowControl digitizer_flow_control_main;
     digitizer_flow_control_main.DigitizerFlowControlStart ();
   }

remove(".plot_data.txt");

}
```

6.59 OutputModule.cpp File Reference

```
#include "DefineGeneral.h"
#include "ConfObject.h"
#include "ApplicationSetup.h"
#include "OutputModule.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
```

6.60 OutputModule.cpp

```

#include "DefineGeneral.h"
#include "ConfObject.h"
#include "ApplicationSetup.h"
#include "OutputModule.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<unistd.h>

OutputModule *
  OutputModule::outputmodule_pInstance = NULL;

OutputModule *
OutputModule::Instance ()
{
```

```cpp
  if (!outputmodule_pInstance)  // Only allow one instance of class to be generated.
    outputmodule_pInstance = new OutputModule ();

  return outputmodule_pInstance;
}

OutputModule::OutputModule ()
{
  OutputModule::OutputModuleSockidOff();
  OutputModule::OutputModuleStdoutOn();

  bzero (buffer, STANDARDBUFFERLIMIT);

  int i;
  for (i = 0; i < 100; i++)
    sockid_array[i] = -1;
}

void
OutputModule::StdOutInsert (const char *string)
{
  bzero (buffer, STANDARDBUFFERLIMIT);
  strncpy (buffer, string, STANDARDBUFFERLIMIT);
}

void
OutputModule::StdOutInsertLex (char *string, int length)
{
  bzero (buffer, STANDARDBUFFERLIMIT);
  strncpy (buffer, string, STANDARDBUFFERLIMIT);
}

void
OutputModule::StdOutPrint ()
{
  output_module_application_setup =
ApplicationSetup::Instance ();

//fprintf(stderr, "Codice dell'ultimo carattere: %d\n", buffer[strlen(buffer)-1]);

  if ((output_module_application_setup->
input_mode == 2
       || output_module_application_setup->
input_mode == 0)
      && output_module_stdout == 1)
    {
      //if (buffer[strlen(buffer)-1] == 10)
      fprintf (stdout, "%s", buffer);
      //else if(buffer[strlen(buffer)-1] != 10){
      //      fprintf(stdout, "%s", buffer);
      //if (strlen(buffer) > 1) fprintf (stdout, "\n");
      //}
    }
}  //void OutputModule::StdOutPrint()

int
OutputModule::TcpUserArrayInsert (int sockid)
{
  int i;
  for (i = 0; i < 100; i++)
    {
      if (sockid_array[i] == -1)
    {
      sockid_array[i] = sockid;
      return 0;
    }
    }
  return 1;
}  //int OutputModule::TcpUserArrayInsert(int sockid)



int
OutputModule::TcpUserArrayDelete (int sockid)
{
  int i;
  for (i = 0; i < 100; i++)
    {
      if (sockid_array[i] == sockid)
    {
      sockid_array[i] = -1;
      return 0;
    }
    }
  return 1;
}  //int OutputModule::TcpUserArrayDelete(int sockid)

```

```

int
OutputModule::TcpUserArraySendStdOut ()
{
  int i;
  output_module_application_setup =
ApplicationSetup::Instance ();

  if (output_module_application_setup->input_mode == 2
      || output_module_application_setup->
input_mode == 1)
    {
      for (i = 0; i < 100; i++)
    {
      if (sockid_array[i] > -1
          && sockid_array[i] == output_module_sockid)
        {
          send (sockid_array[i], buffer, 1000, 0);    //usleep(5000);
        }   //if(sockid_array[i]>-1)
    }   //for(i=0; i<100; i++)
    }

  return 0;
}   //int OutputModule::TcpUserArraySendStdOut(StdOut * stdout_arg)

 void OutputModule::Output(const char * string){

OutputModule::StdOutInsert (string);
OutputModule::StdOutPrint();
OutputModule::TcpUserArraySendStdOut ();

}


void OutputModule::OutputFlex(const char * string, int length)
{
char * new_line_string = (char*)malloc(length+1);
strcpy(new_line_string, string);
new_line_string[length] = new_line_string[length -1];
new_line_string[length -1] = '\n';
OutputModule::StdOutInsertLex(new_line_string, length);
usleep(5000);
OutputModule::StdOutPrint();
OutputModule::TcpUserArraySendStdOut();
free(new_line_string);
}

void OutputModule::OutputModuleStdoutOn(){
output_module_stdout = 1;
}

void OutputModule::OutputModuleStdoutOff(){
output_module_stdout = -1;
}

void OutputModule::OutputModuleSockidOn(int sockid)
{
output_module_sockid = sockid;
}

void OutputModule::OutputModuleSockidOff(){
output_module_sockid = -1;
}
```

6.61 OutputModule.h File Reference

```
#include "DefineGeneral.h"
```

Classes

- class OutputModule

  The OutputModule class provides an useful way to manage the output of the program.

6.62 OutputModule.h

```
#include "DefineGeneral.h"

class OutputModule
{

private:
  OutputModule ();

  static OutputModule *outputmodule_pInstance;

  int output_module_stdout;

  int output_module_sockid;

public:

  ApplicationSetup * output_module_application_setup;

  static OutputModule *Instance ();

  char buffer[STANDARDBUFFERLIMIT];

  void StdOutInsert (const char *string);

  void StdOutInsertLex (char *string, int length);

  void StdOutPrint ();

  void OutputModuleStdoutOn();

  void OutputModuleStdoutOff();

  void OutputModuleSockidOn(int sockid);

  void OutputModuleSockidOff();

  int sockid_array[100];

  int TcpUserArrayInsert (int sockid);

  int TcpUserArrayDelete (int sockid);

  int TcpUserArraySendStdOut ();

  void Output(const char * string);

  void OutputFlex(const char * string, int length);
};
```

6.63 RawData.cpp File Reference

```
#include "DefineGeneral.h"
#include "X742DecodeRoutines.h"
#include "ConfObject.h"
#include "ApplicationSetup.h"
#include "DigitizerErrorObject.h"
#include "LogFile.h"
#include "DigitizerObject.h"
#include "Analizzatore.h"
#include "DigitizerObjectGeneric.h"
#include "RawData.h"
#include "DigitizerStateMachine.h"
#include "OutputModule.h"
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <CAENDigitizer.h>
```

6.64 RawData.cpp

```
#include "DefineGeneral.h"
#include "X742DecodeRoutines.h"
#include "ConfObject.h"
#include "ApplicationSetup.h"
#include "DigitizerErrorObject.h"
#include "LogFile.h"
#include "DigitizerObject.h"
#include "Analizzatore.h"
#include "DigitizerObjectGeneric.h"
#include "RawData.h"
#include "DigitizerStateMachine.h"
#include "OutputModule.h"
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <CAENDigitizer.h>

RawData & RawData::operator= (const RawData & p)
{
  int
    i;
  if (this != &p)
    {
      free (buffer);
      buffer = NULL;
      buffer = (char *) malloc (size);
      for (i = 0; i < p.bsize; i++)
    buffer[i] = p.buffer[i];
      imset = p.imset;
      handle = p.handle;
      bsize = p.bsize;
      size = p.size;
      ret = p.ret;
      eventInfo = p.eventInfo;
      ret_error = p.ret_error;
      Mode = p.Mode;
      file = p.file;
    }
  return *this;
}


RawData::RawData ()
{
  buffer = NULL;
  imset = 0;
}


void
RawData::RawDataSet (DigitizerObjectGeneric digitizer_arg)
{
  digitizer = digitizer_arg;

  handle = digitizer_arg.handle;
  ret = CAEN_DGTZ_MallocReadoutBuffer (handle, &buffer, (uint32_t *) &
size);
  //ret_error.digitizer_error_object_print_error (ret);
}


void
RawData::RawDataDel ()
{
  //ret = CAEN_DGTZ_FreeReadoutBuffer (&buffer);
  free (buffer);
  buffer = NULL;
  //ret_error.digitizer_error_object_print_error (ret);
}


void
RawData::RawDataWriteOnFile (const char *file_arg)
{
  if (bsize > 0)
    {
      FILE *file_size;
      char *file_size_path;
      file_size_path = (char *) malloc (strlen (file_arg) + 3);
      strcpy (file_size_path, file_arg);
      strcat (file_size_path, "sz");
      file = fopen (file_arg, "a");
      file_size = fopen (file_size_path, "a");
      if (file != NULL && file_size != NULL)
    {
      fprintf (file_size, "%d\n", bsize);
      fwrite (buffer, bsize, 1, file);
      fclose (file);
      fclose (file_size);
    }   //if (file != NULL && file_size != NULL)
    }   //if (bsize>0)
}


void
RawData::RawDataWriteOnFile (FILE * file, FILE * file_size)
{
  if (file != NULL && file_size != NULL)
    {
      fprintf (file_size, "%d\n", bsize);
      fwrite (buffer, bsize, 1, file);
    }
}


void
RawData::RawDataWriteDecodeEventOnPlotFile (const char *file_arg)
{

  FILE *file_private_punt;
  file_private_punt = fopen (file_arg, "w");

  ApplicationSetup *application_setup;
  application_setup = ApplicationSetup::Instance ();

  if (file_private_punt != NULL)
    {
    CAEN_DGTZ_UINT8_EVENT_t *Evt8 = NULL;
    CAEN_DGTZ_UINT16_EVENT_t *Evt16 = NULL;
    CAEN_DGTZ_X742_EVENT_t *Evt742 = NULL;
    int max_channels = 0;
    int max_groups = 0;

    int FamilyCode = digitizer.BoardInfo.FamilyCode;

    int FormFactor = digitizer.BoardInfo.FormFactor;

    int i, j, k, o;
    uint32_t numEvents;
    int conta = 0;
    char *evtptr;

        switch(FamilyCode) {
        case CAEN_DGTZ_XX724_FAMILY_CODE:
        case CAEN_DGTZ_XX781_FAMILY_CODE:
        case CAEN_DGTZ_XX720_FAMILY_CODE:
        case CAEN_DGTZ_XX721_FAMILY_CODE:
        case CAEN_DGTZ_XX751_FAMILY_CODE:
        case CAEN_DGTZ_XX761_FAMILY_CODE:
        case CAEN_DGTZ_XX731_FAMILY_CODE:
        switch(FormFactor) {
        case CAEN_DGTZ_VME64_FORM_FACTOR:
        case CAEN_DGTZ_VME64X_FORM_FACTOR:
            max_channels = 8;
            break;
        case CAEN_DGTZ_DESKTOP_FORM_FACTOR:
        case CAEN_DGTZ_NIM_FORM_FACTOR:
            max_channels = 4;
            break;
        }
        break;
        case CAEN_DGTZ_XX730_FAMILY_CODE:
        switch(FormFactor) {
        case CAEN_DGTZ_VME64_FORM_FACTOR:
        case CAEN_DGTZ_VME64X_FORM_FACTOR:
            max_channels = 16;
            break;
        case CAEN_DGTZ_DESKTOP_FORM_FACTOR:
        case CAEN_DGTZ_NIM_FORM_FACTOR:
            max_channels = 8;
            break;
        }
        break;
        case CAEN_DGTZ_XX740_FAMILY_CODE:
        switch(FormFactor) {
        case CAEN_DGTZ_VME64_FORM_FACTOR:
        case CAEN_DGTZ_VME64X_FORM_FACTOR:
            max_channels = 64;
            break;
        case CAEN_DGTZ_DESKTOP_FORM_FACTOR:
        case CAEN_DGTZ_NIM_FORM_FACTOR:
            max_channels = 32;
            break;
        }
        break;
        case CAEN_DGTZ_XX743_FAMILY_CODE:
        switch(FormFactor) {
        case CAEN_DGTZ_VME64_FORM_FACTOR:
        case CAEN_DGTZ_VME64X_FORM_FACTOR:
            max_channels = 16;
            break;
        case CAEN_DGTZ_DESKTOP_FORM_FACTOR:
        case CAEN_DGTZ_NIM_FORM_FACTOR:
            max_channels = 8;
            break;
        }
        break;
        case CAEN_DGTZ_XX742_FAMILY_CODE:
        switch(FormFactor) {
        case CAEN_DGTZ_VME64_FORM_FACTOR:
        case CAEN_DGTZ_VME64X_FORM_FACTOR:
            max_groups = 4;
            break;
        case CAEN_DGTZ_DESKTOP_FORM_FACTOR:
        case CAEN_DGTZ_NIM_FORM_FACTOR:
            max_groups = 2;
            break;
        }
        break;
        //default:
        //assert("This program cannot be used with this digitizer family\n");
        }

      ret = CAEN_DGTZ_GetNumEvents (handle, buffer, bsize, &numEvents);
      conta += numEvents;

      for (i = 0; (unsigned int) i < numEvents; i++)
        {
          ret = CAEN_DGTZ_GetEventInfo (handle, buffer, bsize, i, &eventInfo,
                        &evtptr);

        if (FamilyCode != CAEN_DGTZ_XX742_FAMILY_CODE)
            {

            if (FamilyCode == CAEN_DGTZ_XX721_FAMILY_CODE || FamilyCode == CAEN_DGTZ_XX731_FAMILY_CODE)
                {
                ret = CAEN_DGTZ_DecodeEvent (handle, evtptr, (void **)&Evt8);
                if (application_setup->channel_visualized<max_channels)
                    {
                    for (j=0; j<Evt8->ChSize[application_setup->channel_visualized]; j++)
                        {
                        fprintf(file_private_punt, "%d\n",
                        (int) Evt8->DataChannel[application_setup->channel_visualized][j]);
                        }
                    }
                    CAEN_DGTZ_FreeEvent(handle, (void **) &Evt8);
                }
            else
                {
                ret = CAEN_DGTZ_DecodeEvent (handle, evtptr, (void **)&Evt16);
                if (application_setup->channel_visualized<max_channels)
                    {
                    for (j=0; j<Evt16->ChSize[application_setup->channel_visualized]; j++)
                        {
                        fprintf(file_private_punt, "%d\n",
                        (int) Evt16->DataChannel[application_setup->channel_visualized][j]);
                        }
                    }
                    CAEN_DGTZ_FreeEvent(handle, (void **) &Evt16);
                }
            }

        else
            {
            X742_DecodeEvent (evtptr, (void **)&Evt742);
                    printf("gruppo: %d\ncanale: %d\n", application_setup->channel_visualized / MAX_X742_CHANNEL_SIZE, application_setup->channel_visualized % MAX_X742_CHANNEL_SIZE);
                    if ((application_setup->channel_visualized / MAX_X742_CHANNEL_SIZE) < MAX_X742_GROUP_SIZE)
                        {
                        if (Evt742->GrPresent[application_setup->
channel_visualized / MAX_X742_CHANNEL_SIZE] == 1)
                            {
                            //bzero(stringa, STANDARDBUFFERLIMIT);
                            //snprintf(stringa, STANDARDBUFFERLIMIT, "####GROUP: %d\n", o);
                            //OUTPUT(stringa);
                            //if (application_setup->channel_visualized % MAX_X742_CHANNEL_SIZE <
 MAX_X742_CHANNEL_SIZE)
                                //{
                                //bzero(stringa, STANDARDBUFFERLIMIT);
                                //snprintf(stringa, STANDARDBUFFERLIMIT, "####CHANNEL: %d\n", j);
                                //OUTPUT(stringa);
                                for (k = 0; k < Evt742->DataGroup[application_setup->
channel_visualized / MAX_X742_CHANNEL_SIZE].
                                                    ChSize[application_setup->
channel_visualized % MAX_X742_CHANNEL_SIZE] ; k++)
                                    {
                                    fprintf(file_private_punt, "%d\n",
                                    (int) Evt742->DataGroup[application_setup->
channel_visualized / MAX_X742_CHANNEL_SIZE].
                                                    DataChannel[application_setup->
channel_visualized % MAX_X742_CHANNEL_SIZE][k]);
                                    } //for (k = 0; k < Evt->DataGroup[i].ChSize[j] ; k++)
                                //} //if (application_setup->channel_visualizedMAX_X742_CHANNEL_SIZE)
                            } //if (Evt742->GrPresent[application_setup->channel_visualized /
 MAX_X742_CHANNEL_SIZE] == 1)
                        } //if ((application_setup->channel_visualized / MAX_X742_CHANNEL_SIZE) <
 MAX_X742_GROUP_SIZE)
                CAEN_DGTZ_FreeEvent(handle, (void **) &Evt742);
            } // END OF ELSE

        //fprintf(stderr, "Un segnale: %d\n", (int) Evt->DataChannel[0][0]);

        }   //for (i = 0; (unsigned int) i < numEvents; i++)

        fclose (file_private_punt);
    }   // if (file_private_punt != NULL)
}



void
RawData::RawDataPlot (const char *file_arg, FILE * gnuplot)
{

  uint32_t numEvents = 0;

  ret = CAEN_DGTZ_GetNumEvents (handle, buffer, bsize, &numEvents);

  if (numEvents > 0)
    {
      RawData::RawDataWriteDecodeEventOnPlotFile (file_arg);
      fprintf (gnuplot, "plot '%s'\n", file_arg);
// This line is fundamental in order to send data to the gnuplot program in real-time.
      fflush (gnuplot);
      sleep (1);

    }

}



void
RawData::RawDataRead ()
{
  ret =
    CAEN_DGTZ_ReadData (handle, CAEN_DGTZ_SLAVE_TERMINATED_READOUT_MBLT,
            buffer, (uint32_t *) & bsize);
  //ret_error.digitizer_error_object_print_error (ret);
}



void
RawData::RawDataDecode ()
{

  CAEN_DGTZ_UINT8_EVENT_t *Evt8 = NULL;
  CAEN_DGTZ_UINT16_EVENT_t *Evt16 = NULL;
  CAEN_DGTZ_X742_EVENT_t *Evt742 = NULL;

  char stringa[STANDARDBUFFERLIMIT];
  bzero (stringa, STANDARDBUFFERLIMIT);

  OutputModule *output_module;
  output_module = OutputModule::Instance ();
```

```

int max_channels = 0;
int max_groups = 0;


int FamilyCode = digitizer.BoardInfo.FamilyCode;

int FormFactor = digitizer.BoardInfo.FormFactor;


int i, j, k, o;
uint32_t numEvents;
int conta = 0;
char *evtptr;

  switch(FamilyCode) {
  case CAEN_DGTZ_XX724_FAMILY_CODE:
  case CAEN_DGTZ_XX781_FAMILY_CODE:
  case CAEN_DGTZ_XX720_FAMILY_CODE:
  case CAEN_DGTZ_XX721_FAMILY_CODE:
  case CAEN_DGTZ_XX751_FAMILY_CODE:
  case CAEN_DGTZ_XX761_FAMILY_CODE:
  case CAEN_DGTZ_XX731_FAMILY_CODE:
      switch(FormFactor) {
      case CAEN_DGTZ_VME64_FORM_FACTOR:
      case CAEN_DGTZ_VME64X_FORM_FACTOR:
          max_channels = 8;
          break;
      case CAEN_DGTZ_DESKTOP_FORM_FACTOR:
      case CAEN_DGTZ_NIM_FORM_FACTOR:
          max_channels = 4;
          break;
      }
      break;
  case CAEN_DGTZ_XX730_FAMILY_CODE:
      switch(FormFactor) {
      case CAEN_DGTZ_VME64_FORM_FACTOR:
      case CAEN_DGTZ_VME64X_FORM_FACTOR:
          max_channels = 16;
          break;
      case CAEN_DGTZ_DESKTOP_FORM_FACTOR:
      case CAEN_DGTZ_NIM_FORM_FACTOR:
          max_channels = 8;
          break;
      }
      break;
  case CAEN_DGTZ_XX740_FAMILY_CODE:
      switch(FormFactor) {
      case CAEN_DGTZ_VME64_FORM_FACTOR:
      case CAEN_DGTZ_VME64X_FORM_FACTOR:
          max_channels = 64;
          break;
      case CAEN_DGTZ_DESKTOP_FORM_FACTOR:
      case CAEN_DGTZ_NIM_FORM_FACTOR:
          max_channels = 32;
          break;
      }
      break;
  case CAEN_DGTZ_XX743_FAMILY_CODE:
  switch(FormFactor) {
  case CAEN_DGTZ_VME64_FORM_FACTOR:
  case CAEN_DGTZ_VME64X_FORM_FACTOR:
      max_channels = 16;
      break;
  case CAEN_DGTZ_DESKTOP_FORM_FACTOR:
  case CAEN_DGTZ_NIM_FORM_FACTOR:
      max_channels = 8;
      break;
  }
  break;
  case CAEN_DGTZ_XX742_FAMILY_CODE:
      switch(FormFactor) {
      case CAEN_DGTZ_VME64_FORM_FACTOR:
      case CAEN_DGTZ_VME64X_FORM_FACTOR:
      max_groups = 4;
          break;
      case CAEN_DGTZ_DESKTOP_FORM_FACTOR:
      case CAEN_DGTZ_NIM_FORM_FACTOR:
      max_groups = 2;
          break;
      }
      break;
  //default:
  //assert("This program cannot be used with this digitizer family\n");
  }


  ret = CAEN_DGTZ_GetNumEvents (handle, buffer, bsize, &numEvents);
  conta += numEvents;

  for (i = 0; (unsigned int) i < numEvents; i++)
    {
      ret = CAEN_DGTZ_GetEventInfo (handle, buffer, bsize, i, &
eventInfo,
                           &evtptr);

    if (FamilyCode != CAEN_DGTZ_XX742_FAMILY_CODE)
        {

        if (FamilyCode == CAEN_DGTZ_XX721_FAMILY_CODE || FamilyCode == CAEN_DGTZ_XX731_FAMILY_CODE)
            {
            ret = CAEN_DGTZ_DecodeEvent (handle, evtptr, (void **)&Evt8);
            for (o=0; o<max_channels; o++)
                {
                for (j=0; j<Evt8->ChSize[o]; j++)
                    {
                    //bzero(stringa, STANDARDBUFFERLIMIT);
                        //snprintf(stringa, STANDARDBUFFERLIMIT, "Valore del digitizer: %d\n",
                        //(int) Evt8->DataChannel[o][j]);
                        //output_module->Output(stringa);
                    }
                }
                CAEN_DGTZ_FreeEvent(handle, (void **) &Evt8);
            }
        else
            {
            ret = CAEN_DGTZ_DecodeEvent (handle, evtptr, (void **)&Evt16);
            for (o=0; o<max_channels; o++)
                {
                for (j=0; j<Evt16->ChSize[o]; j++)
                    {
                    //bzero(stringa, STANDARDBUFFERLIMIT);
                        //snprintf(stringa, STANDARDBUFFERLIMIT, "Valore del digitizer: %d\n",
                        //(int) Evt16->DataChannel[o][j]);
                        //output_module->Output(stringa);
                    }
                }
                CAEN_DGTZ_FreeEvent(handle, (void **) &Evt16);
            }
        }
    else
        {
        X742_DecodeEvent (evtptr, (void **)&Evt742);
                for (o=0; o<max_groups; o++)
                        {
                        if (Evt742->GrPresent[o] == 1)
                            {
                            //bzero(stringa, STANDARDBUFFERLIMIT);
                            //snprintf(stringa, STANDARDBUFFERLIMIT, "####GROUP: %d\n", o);
                            //OUTPUT(stringa);
                            for (j=0; j<MAX_X742_CHANNEL_SIZE; j++)
                                {
                                //bzero(stringa, STANDARDBUFFERLIMIT);
                                //snprintf(stringa, STANDARDBUFFERLIMIT, "####CHANNEL: %d\n", j);
                                //OUTPUT(stringa);
                                for (k = 0; k < Evt742->DataGroup[o].ChSize[j] ; k++)
                                    {
                                    //bzero(stringa, STANDARDBUFFERLIMIT);
                                    //snprintf(stringa, STANDARDBUFFERLIMIT, "Valore del digitizer: %d\n",
                                    //(int) Evt742->DataGroup[o].DataChannel[j][k]);
                                    //OUTPUT(stringa);
                                    } //for (k = 0; k < Evt->DataGroup[i].ChSize[j] ; k++)
                                } //for (j=0; j<MAX_X742_CHANNEL_SIZE; j++)
                            } //if (Evt->GrPresent[o] == 1)
                        } //for (o=0; o<MAX_X742_GROUP_SIZE; o++)
            CAEN_DGTZ_FreeEvent(handle, (void **) &Evt742);
        } // END OF ELSE

    //fprintf(stderr, "Un segnale: %d\n", (int) Evt->DataChannel[0][0]);

    }   //for (i = 0; (unsigned int) i < numEvents; i++)
}
```

6.65 RawData.h File Reference

```
#include <assert.h>
#include <stdio.h>
#include <CAENDigitizer.h>
```

Classes

- class RawData

  The RawData class manages the readout from the digitizer.

6.66 RawData.h

```
#include <assert.h>
#include <stdio.h>
#include <CAENDigitizer.h>

class RawData
{

public:

  CAEN_DGTZ_ErrorCode ret;

  CAEN_DGTZ_EventInfo_t eventInfo;

  DigitizerErrorObject ret_error;

  int handle;

  int size;

  CAEN_DGTZ_ReadMode_t Mode;

  int imset;

  FILE *file;

    RawData ();

  int bsize;

  char *buffer;

  DigitizerObjectGeneric digitizer;


  void RawDataSet (DigitizerObjectGeneric digitizer_arg);

  void RawDataDel ();

  void RawDataRead ();

  void RawDataWriteOnFile (const char *file_arg);

  void RawDataWriteOnFile (FILE * file, FILE * file_size);

  void RawDataWriteDecodeEventOnPlotFile (const char *file_arg);

  void RawDataPlot (const char *file_arg, FILE * gnuplot);

  void RawDataDecode ();

    RawData & operator= (const RawData & p);
};
```

6.67 TcpUser.cpp File Reference

```
#include "DefineGeneral.h"
#include "TcpUser.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

6.68 TcpUser.cpp

```

#include "DefineGeneral.h"
#include "TcpUser.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

TcpUser::TcpUser ()
{
  command_sent_by_user = 0;
  bzero (first_parameter, STANDARDBUFFERLIMIT);
  bzero (first_parameter, STANDARDBUFFERLIMIT);
  register_address = -1;
  register_data = -1;
}
```

6.69 TcpUser.h File Reference

`#include "DefineGeneral.h"`

Classes

- class TcpUser

  The TcpUser class provides an useful way to store data about the users of the server.

6.70 TcpUser.h

```

#include "DefineGeneral.h"

class TcpUser
{
public:

  char command_sent_by_user;

  char first_parameter[STANDARDBUFFERLIMIT];

  char second_parameter[STANDARDBUFFERLIMIT];

  int register_address;

  int register_data;

  int user_sockid;

  TcpUser ();
};
```

6.71 X742DecodeRoutines.c File Reference

```
#include "X742DecodeRoutines.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Functions

- int32_t GetNumEvents (char ∗buffer, uint32_t buffsize, uint32_t ∗numEvents)

- int32_t GetEventPtr (char ∗buffer, uint32_t buffsize, int32_t numEvent, char ∗∗EventPtr)
- int32_t X742_DecodeEvent (char ∗evtPtr, void ∗∗Evt)

6.71.1 Function Documentation

6.71.1.1 int32_t GetEventPtr (char ∗ *buffer*, uint32_t *buffsize*, int32_t *numEvent*, char ∗∗ *EventPtr*)

Definition at line 250 of file X742DecodeRoutines.c.

6.71.1.2 int32_t GetNumEvents (char ∗ *buffer*, uint32_t *buffsize*, uint32_t ∗ *numEvents*)

Definition at line 208 of file X742DecodeRoutines.c.

6.71.1.3 int32_t X742_DecodeEvent (char ∗ *evtPtr*, void ∗∗ *Evt*)

Definition at line 301 of file X742DecodeRoutines.c.

6.72 X742DecodeRoutines.c

```
#include "X742DecodeRoutines.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


 static int32_t
getNumberOfBits (uint8_t byte)
{

uint32_t i, count;

count = 0;

for (i = 0; i < 8; i++)
    {

if ((byte >> i) & 0x1)
    count++;

}

return count;

}


static int
V1742UnpackEventGroup (uint32_t group, uint32_t * datain,
               CAEN_DGTZ_X742_GROUP_t * dataout)
{


int i, j, rpnt = 0, wpnt = 0, size1, size2, trg = 0, k;

long samples;

float Time[1024], t0;

float Tsamp;

float vcorr;

uint16_t st_ind = 0;

uint32_t freq;

float wave_tmp[1024];


freq = (datain[0] >> 16) & 0x3;

switch (freq)
```

```
    {

case CAEN_DGTZ_DRS4_2_5GHz:

Tsamp = (float) ((1.0 / 2500.0) * 1000.0);

break;

case CAEN_DGTZ_DRS4_1GHz:

Tsamp = (float) ((1.0 / 1000.0) * 1000.0);

break;

default:

Tsamp = (float) ((1.0 / 5000.0) * 1000.0);

break;

}


st_ind = (uint16_t) ((datain[0] >> 20) & 0x3FF);

size1 = datain[0] & 0xFFF;

if ((trg = (datain[0] >> 12) & 0x1) == 1)

size2 = (datain[0] >> 3) & 0x1FF;

  else

size2 = 0;


dataout->TriggerTimeTag = datain[size1 + size2 + 1] & 0x3FFFFFFF;


samples = ((long) (size1 / 3));


while (rpnt < size1)
    {


switch (rpnt % 3)
    {

case 0:

dataout->DataChannel[0][wpnt] = (float) (datain[rpnt + 1] & 0x00000FFF);    /* S0[11:0] - CH0 */

dataout->DataChannel[1][wpnt] = (float) ((datain[rpnt + 1] & 0x00FFF000) >> 12);    /* S0[11:0] - CH1 */

dataout->DataChannel[2][wpnt] = (float) ((datain[rpnt + 1] & 0xFF000000) >> 24);    /* S0[ 7:0] - CH2 */

break;

case 1:

dataout->DataChannel[2][wpnt] +=
        (float) ((datain[rpnt + 1] & 0x0000000F) << 8);

dataout->DataChannel[3][wpnt] = (float) ((datain[rpnt + 1] & 0x0000FFF0) >> 4); /* S0[11:0] - CH3 */

dataout->DataChannel[4][wpnt] = (float) ((datain[rpnt + 1] & 0x0FFF0000) >> 16);    /* S0[11:0] - CH4 */

dataout->DataChannel[5][wpnt] = (float) ((datain[rpnt + 1] & 0xF0000000) >> 28);    /* S0[3:0]  - CH5 */

break;

case 2:

dataout->DataChannel[5][wpnt] +=
        (float) ((datain[rpnt + 1] & 0x000000FF) << 4);

dataout->DataChannel[6][wpnt] = (float) ((datain[rpnt + 1] & 0x000FFF00) >> 8); /* S0[11:0] - CH6 */

dataout->DataChannel[7][wpnt] = (float) ((datain[rpnt + 1] & 0xFFF00000) >> 20);    /* S0[11:0] - CH7 */

wpnt++;

break;

}

rpnt++;

}

rpnt++;

for (k = 0; k < 8; k++)
    dataout->ChSize[k] = wpnt;

wpnt = 0;


for (i = 0; i < size2; i++)
    {

switch (i % 3)
    {

case 0:

dataout->DataChannel[8][wpnt] = (float) (datain[rpnt + i] & 0x00000FFF);     /* S0 - CH8 */

dataout->DataChannel[8][++wpnt] = (float) ((datain[rpnt + i] & 0x00FFF000) >> 12);  /* S1 - CH8 */

dataout->DataChannel[8][++wpnt] = (float) ((datain[rpnt + i] & 0xFF000000) >> 24);  /* S2[ 7:0] - CH8 */

break;

case 1:

dataout->DataChannel[8][wpnt] +=
        (float) ((datain[rpnt + i] & 0x0000000F) << 8);

dataout->DataChannel[8][++wpnt] = (float) ((datain[rpnt + i] & 0x0000FFF0) >> 4);   /* S3 - CH8 */

dataout->DataChannel[8][++wpnt] = (float) ((datain[rpnt + i] & 0x0FFF0000) >> 16);  /* S4 - CH8 */

dataout->DataChannel[8][++wpnt] = (float) ((datain[rpnt + i] & 0xF0000000) >> 28);  /* S5[3:0]  - CH8 */

break;

case 2:

dataout->DataChannel[8][wpnt] += (float) ((datain[rpnt + i] & 0x000000FF) << 4);    /* S5[11:4] - CH8 */

dataout->DataChannel[8][++wpnt] = (float) ((datain[rpnt + i] & 0x000FFF00) >> 8);   /* S6[11:0] - CH8 */

dataout->DataChannel[8][++wpnt] = (float) ((datain[rpnt + i] & 0xFFF00000) >> 20);  /* S7[11:0] - CH8 */

wpnt++;

break;

}

}

dataout->ChSize[8] = wpnt;

dataout->StartIndexCell = (uint16_t) st_ind;

return (size1 + size2 + 2);

}



int32_t GetNumEvents (char *buffer, uint32_t buffsize,
             uint32_t * numEvents)
{

uint32_t i = 0, evtSize;

int ret;

int32_t counter = -1;

if ((buffsize == 0) || (buffer == NULL))
    {

*numEvents = 0;

return 0;

}

if (buffsize < EVENT_HEADER_SIZE)
```

```
    return -1;

  do
    {

counter++;

evtSize = *(long *) (buffer + i) & 0x0FFFFFFF;

i += (uint32_t) (evtSize * 4);

}
  while ((i + EVENT_HEADER_SIZE) < buffsize);

*numEvents = counter + 1;

return 0;

}



int32_t GetEventPtr (char *buffer, uint32_t buffsize, int32_t numEvent,
            char **EventPtr)
{

uint32_t i = 0;

int32_t counter = -1;

int ret;

int evtSize;


if ((buffer == NULL) || (buffsize < EVENT_HEADER_SIZE))
    return -1;

  do
    {

counter++;

evtSize = *(long *) (buffer + i) & 0x0FFFFFFF;

if (counter == numEvent)
    {

if ((i + (uint32_t) evtSize) < buffsize)
        {

*EventPtr = (buffer + i);

return 0;

}

      else
        return -1;

}

i += (uint32_t) (evtSize * 4);

}
  while ((i + EVENT_HEADER_SIZE) < buffsize);

return -1;

}



int32_t X742_DecodeEvent (char *evtPtr, void **Evt)
{

CAEN_DGTZ_X742_EVENT_t * Event;

uint32_t * buffer;

char chanMask;

uint32_t j, g, size;

uint32_t * pbuffer;

uint32_t eventSize;
```

```

int evtSize, h;


evtSize = *(long *) evtPtr & 0x0FFFFFFF;

chanMask = *(long *) (evtPtr + 4) & 0x0000000F;

evtPtr += EVENT_HEADER_SIZE;

buffer = (uint32_t *) evtPtr;

pbuffer = (uint32_t *) evtPtr;

eventSize = (evtSize * 4) - EVENT_HEADER_SIZE;

if (eventSize == 0)
    return -1;

Event =
    (CAEN_DGTZ_X742_EVENT_t *) malloc (sizeof (CAEN_DGTZ_X742_EVENT_t));

if (Event == NULL)
    return -1;

memset (Event, 0, sizeof (CAEN_DGTZ_X742_EVENT_t));

for (g = 0; g < X742_MAX_GROUPS; g++)
    {

if ((chanMask >> g) & 0x1)
    {

for (j = 0; j < MAX_X742_CHANNEL_SIZE; j++)
        {

Event->DataGroup[g].DataChannel[j] =
        (float *) malloc (X742_FIXED_SIZE * sizeof (float));

if (Event->DataGroup[g].DataChannel[j] == NULL)
        {

for (h = j - 1; h > -1; h++)
            free (Event->DataGroup[g].DataChannel[h]);

return -1;

}

}

size = V1742UnpackEventGroup (g, pbuffer, &(Event->DataGroup[g]));

pbuffer += size;

Event->GrPresent[g] = 1;

}

      else
    {

Event->GrPresent[g] = 0;

for (j = 0; j < MAX_X742_CHANNEL_SIZE; j++)
        {

Event->DataGroup[g].DataChannel[j] = NULL;

}

}

}

*Evt = Event;

return 0;

}


```

6.73 X742DecodeRoutines.h File Reference

```
#include <CAENDigitizer.h>
```

Macros

- #define EVENT_HEADER_SIZE 0x10
- #define X742_MAX_GROUPS 0x04
- #define X742_FIXED_SIZE 0x400

Functions

- int32_t GetNumEvents (char ∗buffer, uint32_t buffsize, uint32_t ∗numEvents)
- int32_t GetEventPtr (char ∗buffer, uint32_t buffsize, int32_t numEvent, char ∗∗EventPtr)
- int32_t X742_DecodeEvent (char ∗evtPtr, void ∗∗Evt)

6.73.1 Macro Definition Documentation

6.73.1.1 #define EVENT_HEADER_SIZE 0x10

Definition at line 3 of file X742DecodeRoutines.h.

6.73.1.2 #define X742_FIXED_SIZE 0x400

Definition at line 7 of file X742DecodeRoutines.h.

6.73.1.3 #define X742_MAX_GROUPS 0x04

Definition at line 5 of file X742DecodeRoutines.h.

6.73.2 Function Documentation

6.73.2.1 int32_t GetEventPtr (char ∗ *buffer,* uint32_t *buffsize,* int32_t *numEvent,* char ∗∗ *EventPtr*)

Definition at line 250 of file X742DecodeRoutines.c.

6.73.2.2 int32_t GetNumEvents (char ∗ *buffer,* uint32_t *buffsize,* uint32_t ∗ *numEvents*)

Definition at line 208 of file X742DecodeRoutines.c.

6.73.2.3 int32_t X742_DecodeEvent (char ∗ *evtPtr,* void ∗∗ *Evt*)

Definition at line 301 of file X742DecodeRoutines.c.

6.74 X742DecodeRoutines.h

```
#include <CAENDigitizer.h>

#define EVENT_HEADER_SIZE   0x10

#define X742_MAX_GROUPS   0x04

#define X742_FIXED_SIZE   0x400

/******************************************************************************
* GetNumEvents(char *buffer, uint32_t buffsize, uint32_t *numEvents)
* Gets current number of event stored in the acquisition buffer
*
* [IN] buffer     : Address of the acquisition buffer
* [IN] bufferSize : Size of the data stored in the acquisition buffer
* [OUT] numEvents : Number of events stored in the acquisition buffer
*                 : return  0 = Success;
*******************************************************************************/
  int32_t GetNumEvents (char *buffer, uint32_t buffsize,
            uint32_t * numEvents);


/******************************************************************************
* GetEventPtr(char *buffer, uint32_t buffsize, int32_t numEvent, char **EventPtr)
* Retrieves the event pointer of a specified event in the acquisition buffer
*
* [IN] buffer     : Address of the acquisition buffer
* [IN] bufferSize : Acquisition buffer size (in samples)
* [IN] numEvents  : Number of events stored in the acquisition buffer
* [OUT] EventPtr  : Pointer to the requested event in the acquisition buffer
*                 : return  0 = Success;
*******************************************************************************/
  int32_t GetEventPtr (char *buffer, uint32_t buffsize, int32_t numEvent,
               char **EventPtr);


/*******************************************************************************
* X742_DecodeEvent(char *evtPtr, void **Evt)
* Decodes a specified event stored in the acquisition buffer writing data in Evt memory
* Once used the Evt memory MUST be deallocated by the caller!
*
* [IN]  EventPtr : pointer to the requested event in the acquisition buffer (MUST BE NULL)
* [OUT] Evt      : event structure with the requested event data
*                : return  0 = Success;
*******************************************************************************/
  int32_t X742_DecodeEvent (char *evtPtr, void **Evt);

```

Index