blue PiraT Client Library 4.3.3

Bedienungsanleitung

Generated by Doxygen 1.4.0

Die Feb 12 2013 13:03:07

Inhaltsverzeichnis

Kapitel 1

Bedienungsanleitung - blue PiraT Client Library 4.3.3

Die vorliegende Bedienungsanleitung basiert auf der Client-Library 4.3.3

1.1 Architektur

Abbildung 1 zeigt eine vereinfachte Architektur einer Benutzeranwendung und die Integration des blue PiraT Systems über die blue PiraT Client Library.

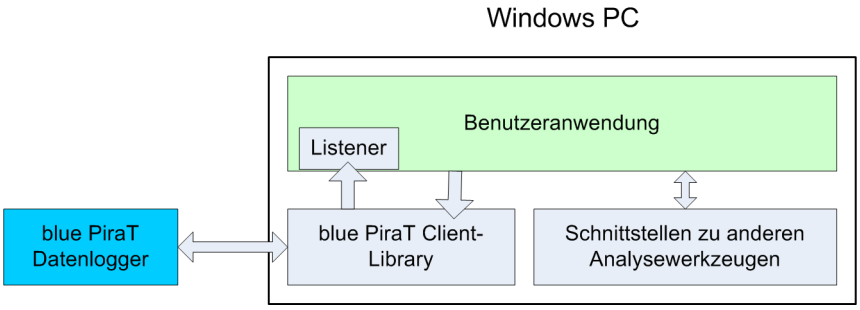


Abbildung 1.1: Einbindung der Client Library

Die Library ist in C++ geschrieben. Die Benutzeranwendung verwendet C++ Objekte der Library und ruft deren Methoden auf. Dabei sind die Aufrufe grundsätzlich blockierend, d.h. sie kehren erst nach ihrer Vollendung zum Aufrufer zurück. Während dieser Zeit ist es über Listener--Mechanismen möglich, dass die Client-Library Funktionen der Benutzeranwendung aufruft, z.B. für Status- oder Fortschrittsrückmeldungen.

1.1.1 Fehlerbehandlung - Listenermechanismus

Die Library kommuniziert mit der aufrufenden Applikation auf zwei Arten. Alle Rückmeldungen die den Programmfluss nicht abbrechen, werden über den Aufruf von Listenerfunktionen mitgeteilt - dazu zählen Status- und Fortschrittsangaben, Warnungen, User-Informationen und Konver-

tierungsfehler. Dafür muss die Benutzeranwendung von der Listener Klasse `BluePiratClient-Listener` abgeleitet sein und deren Funktionen soweit gewünscht implementieren. Alle Fehler die das Fortsetzen der Datenübertragung unmöglich machen, werden der Applikation über Exceptions mitgeteilt. Daraus folgt, dass alle Aufrufe von Library-Funktionen in einen oder mehrere try-catch-Block(e) eingeschlossen werden müssen. Die Exceptions sind vom Typ `BluePirat-ClientException`.

1.2 Funktionsumfang

Die vorliegende Version der blue PiraT Client Library basiert auf der Client-Version 4.3.3 Es wird die Funktionalität der Datenübertragungs-Software und der Konfigurations-Software mit eingeschränktem Funktionsumfangunterstützt. Die Klasse `BluePiratClient` bildet die Schnittstelle zur Library.

1.3 Compiler/Linker

Die Library wurde mit Microsoft Visual Studio unter Verwendung der Laufzeitbibliothek Multithreaded-DLL erstellt und steht für MSVC++ 7.1 (2003), MSVC++ 8.0 (2005) und MSVC++ 9.0 (2008) zur Verfügung. Stellen Sie sicher, dass Ihr Projekt ebenfalls mit dieser Laufzeitbibliothek erstellt wird, da es sonst zu Fehlern kommen kann. Bitte linken Sie zu Ihrer Debug/Release-Version die entsprechende Library. Debug-Libraries sind mit der Endung "_d" im Dateinamen gekennzeichnet. Die Library ist nicht thread-safe.

1.4 Demo-Projekt

Im Installationsverzeichnis bzw. im Startmenü finden Sie zwei Demo-Projekte zur Verwendung der blue PiraT Client Library.

Beispiel zur Verwendung der Library:

```
//
      ******************************************************************************
//
      ******************************************************************************
// $Id$
//
      ******************************************************************************
//
      ******************************************************************************

#include <iostream>
#include <conio.h>
#include <direct.h>

#include "BluePiratClient.hh"
#include "ClientConfiguration.hh"
#include "BluePiratClientException.hh"
#include "Channel.hh"

using namespace std;
using namespace bp;
```

```
//====================================================================
// Dies ist ein Minimalbeispiel zur Verwendung der blue PiraT Client Library.
// Es werden alle Tracedaten seit dem letzten Aufstarten des angeschlossenen
// Datenloggers in ein Verzeichnis "OfflineData" im temporären Verzeichnis
       c:\Temp\bpc-test
// herunter geladen.
//====================================================================
void main()
{
    try
    {
        _mkdir("C:/Temp/bpc-test/");

        // Konstruktor
        BluePiratClient bpc("C:/Temp/bpc-test/");

        /* Soll ein bestimmter Logger ausgelesen werden, muss die Funktion
       BluePiratClient::setLoggerIP()
           aufgerufen werden. Eine Liste mit allen im Netzwerk befindlichen
       Loggern liefert
           BluePiratClient::getListOfLoggersInNetwork()
           Wird keine IP-Adresse gesetzt, versucht sich die Library mit dem
       Standard-Logger
           zu verbinden. */

        // bpc.setLoggerIP(ip);

        // Initialisierung der Datenübertragung
        bpc.initTransfer();

        EventContainer events = bpc.getEventContainer();
        int i;
        bool lastStartupFound = false;
        for (i = events.getNumEntries() - 1; i >= 0; i--)
        {
            // selektiert nur den letzten Startup im Container zum download
            if (!lastStartupFound && events.getEntry(i).getEventType() ==
      STARTUP)
            {
                events.getEntry(i).setTransferFlag(true);
                lastStartupFound = true;
            }
            else
                events.getEntry(i).setTransferFlag(false);
        }

        // Konfiguration der Konvertierung
        ClientConfiguration config;
        config.setChannelToFormat(Channel::CAN, 0, CANOE);
        config.setChannelToFormat(Channel::CAN, 1, CANOE);
        config.setChannelToFormat(Channel::MOST_CTRL, 0, OP2);

        config.setTargetDirectory("C:/Temp/bpc-test/");
        config.setNameOfTester("Tester");
        config.useLongFileNameFormat(true);

        // Übergeben der Konfiguratio an die Library
        bpc.setClientConfiguration(config);

        // Herunterladen und konvertieren der Tracedaten
        bpc.downloadAndConvertData(events);
    }
    catch(BluePiratClientException exception)
    {
        cout << "BPC ERROR: " << exception.getMessage() << endl;
        cout << "Druecken Sie 'q' zum Beenden" << endl;
        char in = getch();
```

```
        while (in != 'q' && in != 'Q')//cin >> ret;
            in = getch();
        cin.clear();

        return;
    }
}
```

Kapitel 2

Modul-Verzeichnis

2.1 Module

Hier folgt die Aufzählung aller Module:

Kapitel 3

Verzeichnis der Namensbereiche

3.1 Liste aller Namensbereiche

Liste aller Namensbereiche mit Kurzbeschreibung:

Kapitel 4

Klassen-Verzeichnis

4.1 Klassenhierarchie

Die Liste der Ableitungen ist -mit Einschränkungen- alphabetisch sortiert:

Kapitel 5

Klassen-Verzeichnis

5.1 Auflistung der Klassen

Hier folgt die Aufzählung aller Klassen, Strukturen, Varianten und Schnittstellen mit einer Kurzbeschreibung:

Kapitel 6

Datei-Verzeichnis

6.1 Auflistung der Dateien

Hier folgt die Aufzählung aller Dateien mit einer Kurzbeschreibung:

Kapitel 7

Modul-Dokumentation

7.1 Data Download and Conversion Classes

Klassen

- class **BluePiratClient**
  Interface Class for the blue PiraT Client library.
- class **ClientConfiguration**
  Class to configure the client library settings.
- class **EventContainer**
  Class to store the entries of the event file.
- class **EventContainerEntry**
  Class to encapsulate one Events file entry.
- class **TimeSpanContainer**
  Class to store a number of TimeSpans.
- class **TimeSpan**
  Class to store a time span.
- class **Channel**
  Class to represent one channel.

Aufzählungen

- enum **FormatId** {
  NOTRANSFER = 0, **TMASC** = 1, **OP2** = 2, **CANOE** = 3,
  STA = 4, **GNLOG_SINGLE** = 5, **TCLOG** = 6, **TCLOG_TS** = 7,
  RAW_SERIAL = 8, **IMG** = 10, **TM** = 11, **CANCORDER** = 12,
  GNLOG_SHARED = 13, **MPEG4_BLOCKS** = 14, **APN** = 15, **ASCHEX** = 16,
  IPOD_COMMAND = 17, **WAV** = 19, **TCPDUMP** = 21, **MPEG4_JOINED_HQ** = 22,
  BLF = 23, **MDF** = 24, **DLT_BMW** = 25, **ETH_RAW** = 26,
  INVALID = 0xFF }

7.1.1 Dokumentation der Aufzählungstypen

7.1.1.1 enum FormatId

Holds all possible file formats. A new format has to be added here and initialised in TransferFormatProperties::initialise().

Aufzählungswerte:

NOTRANSFER No transfer.

TMASC Telemotive ASCII format.

OP2 MOST OPformat.

CANOE CANOE ASCII format.

STA Serial Trace Analyser format.

GNLOG_SINGLE GNLog format.

TCLOG Trace Client format.

TCLOG_TS Trace Client format plus time stamps.

RAW_SERIAL Raw Serial Format.

IMG DataAnalyser IMG format.

TM Telemotive binary format.

CANCORDER CANCorder format.

GNLOG_SHARED GNlog format for ethernet and serial trace.

MPEG4_BLOCKS Video format, transfers the blocks from the logger.

APN APN serial format.

ASCHEX ASCII format that writes binary data as hex values.

IPOD_COMMAND ASCII representation of ipod commands.

WAV Wave PCM Format, use this for MOST Sync.

TCPDUMP TCP dump (∗.pcap) format.

MPEG4_JOINED_HQ Video format join blocks with frame copying (high quality)

BLF Binary Logging Format - Vector Informatic.

MDF Measurement data format - Vector Informatic.

DLT_BMW DLT Data Trace and Logging - BMW Autosar.

ETH_RAW

INVALID Ethernet RAW format.

7.2 Logger Configuration Classes

Klassen

- class **BluePiratClient**

  Interface Class for the blue PiraT Client library.
- class **LoggerConfiguration**

  Class to configure the data logger via client library.
- class **CanFilterProperties**

  Property class to store the filtered CAN IDs.
- class **CanProperties**

  Property class to store the CAN properties of a CAN interface.
- class **CascadingProperties**

  This class stores the properties for the logger's cascading feature.
- class **DataStorageProperties**

  The property class stores settings that consider the storage and protection of recorded trace data.
- class **FlexrayGeneralProperties**

  This class enables to read/write flexray general properries from/to the ConfigConatiner.
- class **FlexrayProperties**

  This class enables to read/write flexray properries from/to the ConfigConatiner.
- class **GeneralProperties**

  Property class to store general logger settings.
- class **LinProperties**

  Property class to store the LIN settings.
- class **MarkerProperties**

  Property class to store the CAN messages related to the markers.
- class **MarkerProtectionProperties**

  Property class to store the length of the data block to protect around a marker time stamp.
- class **MostFilterProperties**

  Property class to store most filter.
- class **MostProperties**

  Property class to store the MOST configuratio settings.
- class **NetworkConfigProperties**

  Property class to store the dhcp settings.
- class **SerialProperties**

  Property class to store the serial interface settings.
- class **SleepProperties**

  Property class to store the power management settings.
- class **RCVoiceProperties**

  Property class to store the RCVoice configuration settings.

Kapitel 8

Dokumentation der Namensbereiche

8.1 bp-Namensbereichsreferenz

Klassen

- class **BluePiratClient**
  Interface Class for the blue PiraT Client library.
- class **BluePiratClientListener**
- class **ClientConfiguration**
  Class to configure the client library settings.
- class **EventContainerListener**
- class **EventContainer**
  Class to store the entries of the event file.
- class **EventContainerEntry**
  Class to encapsulate one Events file entry.
- class **TimeSpanContainer**
  Class to store a number of TimeSpans.
- class **TimeSpan**
  Class to store a time span.
- class **Channel**
  Class to represent one channel.
- class **TransferApplicationListener**
  Class to listen the TransferApplication.
- class **TransferApplicationException**
  Class for transfer application errors, warnings, info.
- struct **LoggerInNetwork**
- struct **VersionsInfoEntry**
- struct **CANChannelClockFrequency**
  Type definition for the Configuration Tool.
- struct **ChangedDBPathChannel**
- struct **InterfaceParametersStruct**
  Struct for interface parameters.
- class **LoggerConfiguration**

Class to configure the data logger via client library.

- class **CanFilterProperties**

  Property class to store the filtered CAN IDs.
- class **CanProperties**

  Property class to store the CAN properties of a CAN interface.
- class **CascadingProperties**

  This class stores the properties for the logger's cascading feature.
- class **DataStorageProperties**

  The property class stores settings that consider the storage and protection of recorded trace data.
- class **GeneralProperties**

  Property class to store general logger settings.
- class **LinProperties**

  Property class to store the LIN settings.
- class **MarkerProperties**

  Property class to store the CAN messages related to the markers.
- class **MarkerProtectionProperties**

  Property class to store the length of the data block to protect around a marker time stamp.
- class **MostFilterProperties**

  Property class to store most filter.
- class **MostProperties**

  Property class to store the MOST configuratio settings.
- class **NetworkConfigProperties**

  Property class to store the dhcp settings.
- class **SerialProperties**

  Property class to store the serial interface settings.
- class **SleepProperties**

  Property class to store the power management settings.
- class **RCVoiceProperties**

  Property class to store the RCVoice configuration settings.
- class **TimeZone**
- class **BluePiratClientException**

  Class for blue PiraT Client library errors.

Aufzählungen

- enum **EventType** {
  HEADLINE = 0, **STARTUP** = 1, **MARKER** = 2, **SHUTDOWN** = 3,
  DATAERASED = 4, **SLAVEOFFSET** = 5, **SLAVETOMASTER** = 6, **INFO** = 7,
  DATADOWNLOAD = 8, **ERROR_EVENT** = 9, **SETTIME** = 10, **NEWTIME** = 11,
  UNKNOWN_EVENT = 99 }

  Event types used by application and user interface, HEADLINE only for GUI.
- enum **OverwritingResponse** {
  YES = 0, **NO**, **YES_TO_ALL**, **NO_TO_ALL**,
  ABORT }
- enum **LoggerStatus** { **LOGGER_FOUND**, **LOGGER_NOT_FOUND** }

  Logger status.

- enum **CONFIG_IO_MODE** {
  CONFIG_WRITE_TO_LOGGER, **CONFIG_READ_FROM_LOGGER**, **CONFIG_SAVE_LOCALLY**, **CONFIG_LOAD_LOCALLY**,
  CONFIG_PASSWORD_UPDATE, **CONFIG_DATE_TIME_UPDATE** }

  Type definition for the Configuration Tool.
- enum **FirmwareFeatures** {
  NONE_FIRMWARE_FEATURE, **CAN_ACKNOWLEDGE_MODE**, **CAN_BIT_TIMING_MODE**, **LOGGER_CASCADING_FEATURE**,
  MOST_FEATURE, **MOST_TRACE_ASYNC_CHANNEL_MODE**, **MOST_ASYNC_ARBITRATION_VALUE_MODE**, **MOST_DATA_REGENERATION_MODE**,
  RECORDING_MOST_ERROR_MESSAGES, **MOST_TRACE_SYNC_CHANNEL_MODE**, **MOST50_TRACE_SYNC_CHANNEL_MODE**, **LOGGER_AUTOMATIC_SHUTDOWN_DEACTIVATED_MODE**,
  GNLOG_OVER_ETHERNET_FEATURE, **ETHERNET_LOGGING_EXTENDED_IP_CONFIG**, **CAN_TRIGGER_MASKING_FEATURE**, **CAMERA_FEATURE**,
  LIN_FEATURE, **RC_MONITOR_FEATURE**, **COMPLEX_TRIGGER_FEATURE**, **CAMERA_EXTENTION_FEATURE**,
  MULTIPLE_CAMERA_SUPPORT_FEATURE, **TRACE_FILE_COMPRESSION_FEATURE**, **EXTENDED_TRIGGER_CONFIG_FEATURE**, **TRIGGER_TYPE_CONFIG_FEATURE**,
  FLEXRAY_FEATURE, **ETHERNET_LOGGING_PORT**, **RC_VOICE_FEATURE**, **CASCADING_CHANNEL_OFFSET_EXTENSION**,
  NETWORK_CONFIG_FEATURE, **MOST150_FEATURE**, **MOST150_FILTER_FEATURE**, **CAMERA_PASSWORD_FEATURE**,
  ETHERNET_RAW_UTF8_LOGGING_FEATURE, **INTERFACE_NAME_MAPPING_FEATURE**, **ETHERNET_TIMEOUT_FEATURE**, **ETHERNET_UDP_SERVER_FEATURE**,
  MOST50_FEATURE, **MOST50_FILTER_FEATURE**, **DLT_BMW_LOGGING_FEATURE**, **ETHERNET_VLAN_FEATURE**,
  MOST_TRAINING }

  Type definition for the Configuration Tool.
- enum **LoggerFeatures** {
  NONE_LOGGER_FEATURE, **NUM_CAN_CHANNELS**, **NUM_SW_CHANNELS**, **NUM_TRANSCEIVER_TYPES**,
  ETHERNET_SWITCH, **NUM_MOST25_CHANNELS** }
- enum **FeaturesType** { **FIRMWARE_CAPABILITIES**, **DEVICE_CAPABILITIES**, **LICENSES**, **FLEXRAY_CONFIG** }

  Type definition for the Configuration Tool.
- enum **ConfigType** { **OCEAN_INI**, **TRIGGER_INI** }
- enum **LocalLanguage** { **GERMAN**, **US_ENGLISH** }
- enum **LowerBoundType** { **PRETIME**, **LAST_STARTUP**, **UNKNOWN_LB** }
- enum **UpperBoundType** {
  POSTTIME, **NEXT_SHUTDOWN**, **NEXT_MARKER_INFO**, **NEXT_TEXT_INFO**,
  UNKNOWN_UB }

8.1.1 Dokumentation der Aufzählungstypen

8.1.1.1 enum EventType

Event types used by application and user interface, HEADLINE only for GUI.

Those constants represent the possible events.

Aufzählungswerte:

HEADLINE only for GUI
STARTUP Datalogger started tracing.
MARKER User set a marker.
SHUTDOWN Datalogger shutdown.
DATAERASED User deleted all data from logger.
SLAVEOFFSET Time Offset for salve logger.
SLAVETOMASTER Master shutdown when slave still active.
INFO
DATADOWNLOAD Info for ROE light messages.
ERROR_EVENT Data download with client. corrupt event
SETTIME
NEWTIME Time was set. New time that was set
UNKNOWN_EVENT

8.1.1.2 enum OverwritingResponse

Aufzählungswerte:

YES
NO
YES_TO_ALL
NO_TO_ALL
ABORT

8.1.1.3 enum LoggerStatus

Logger status.

Possible responses from **BluePiratClient::detectLogger** (S. 35)

Aufzählungswerte:

LOGGER_FOUND
LOGGER_NOT_FOUND

8.1.1.4 enum CONFIG_IO_MODE

Type definition for the Configuration Tool.

Aufzählungswerte:

CONFIG_WRITE_TO_LOGGER
CONFIG_READ_FROM_LOGGER

CONFIG_SAVE_LOCALLY
CONFIG_LOAD_LOCALLY
CONFIG_PASSWORD_UPDATE
CONFIG_DATE_TIME_UPDATE

8.1.1.5 enum FirmwareFeatures

Type definition for the Configuration Tool.

Aufzählungswerte:

NONE_FIRMWARE_FEATURE
CAN_ACKNOWLEDGE_MODE
CAN_BIT_TIMING_MODE
LOGGER_CASCADING_FEATURE
MOST_FEATURE
MOST_TRACE_ASYNC_CHANNEL_MODE
MOST_ASYNC_ARBITRATION_VALUE_MODE
MOST_DATA_REGENERATION_MODE
RECORDING_MOST_ERROR_MESSAGES
MOST_TRACE_SYNC_CHANNEL_MODE
MOST50_TRACE_SYNC_CHANNEL_MODE
LOGGER_AUTOMATIC_SHUTDOWN_DEACTIVATED_MODE
GNLOG_OVER_ETHERNET_FEATURE
ETHERNET_LOGGING_EXTENDED_IP_CONFIG
CAN_TRIGGER_MASKING_FEATURE
CAMERA_FEATURE
LIN_FEATURE
RC_MONITOR_FEATURE
COMPLEX_TRIGGER_FEATURE
CAMERA_EXTENTION_FEATURE
MULTIPLE_CAMERA_SUPPORT_FEATURE
TRACE_FILE_COMPRESSION_FEATURE
EXTENDED_TRIGGER_CONFIG_FEATURE
TRIGGER_TYPE_CONFIG_FEATURE
FLEXRAY_FEATURE
ETHERNET_LOGGING_PORT
RC_VOICE_FEATURE
CASCADING_CHANNEL_OFFSET_EXTENSION
NETWORK_CONFIG_FEATURE
MOST150_FEATURE
MOST150_FILTER_FEATURE

CAMERA_PASSWORD_FEATURE
ETHERNET_RAW_UTF8_LOGGING_FEATURE
INTERFACE_NAME_MAPPING_FEATURE
ETHERNET_TIMEOUT_FEATURE
ETHERNET_UDP_SERVER_FEATURE
MOST50_FEATURE
MOST50_FILTER_FEATURE
DLT_BMW_LOGGING_FEATURE
ETHERNET_VLAN_FEATURE
MOST_TRAINING

8.1.1.6 enum LoggerFeatures

Aufzählungswerte:

NONE_LOGGER_FEATURE
NUM_CAN_CHANNELS
NUM_SW_CHANNELS
NUM_TRANSCEIVER_TYPES
ETHERNET_SWITCH
NUM_MOST25_CHANNELS

8.1.1.7 enum FeaturesType

Type definition for the Configuration Tool.

Aufzählungswerte:

FIRMWARE_CAPABILITIES
DEVICE_CAPABILITIES
LICENSES
FLEXRAY_CONFIG

8.1.1.8 enum ConfigType

Aufzählungswerte:

OCEAN_INI
TRIGGER_INI

8.1.1.9 enum LocalLanguage

Aufzählungswerte:

GERMAN

US_ENGLISH

8.1.1.10 enum LowerBoundType

All possible boundary definitions for the lower time span boundary when selecting a marker for download/conversion. Don't use UNKNOWN_LB

Aufzählungswerte:

PRETIME Defines that a fix time span before the marker should be selected.

LAST_STARTUP Defines that the previous startup marks the start of the marker time span.

UNKNOWN_LB

8.1.1.11 enum UpperBoundType

All possible boundary definitions for the upper time span boundary when selecting a marker for download/conversion. Don't use UNKNOWN_UB

Aufzählungswerte:

POSTTIME Defines that a fix time span after the marker should be selected.

NEXT_SHUTDOWN Defines that the next shutdown marks the end of the marker time span.

NEXT_MARKER_INFO Defines that the next marker or info event marks the end of the time span.

NEXT_TEXT_INFO Defines that the next info event with a specified info text marks the end of the time span.

UNKNOWN_UB

Kapitel 9

Klassen-Dokumentation

9.1 BluePiratClient Klassenreferenz

Interface Class for the blue PiraT Client library.

Öffentliche Methoden

- **BluePiratClient** (const char ∗tempDirectory, const char ∗loggerIP="")
  Constructor, needs the path to a temporary directory as parameter.
- ∼**BluePiratClient** ()
  Destructor.
- void **setLoggerIP** (std::string ip)
  *Sets the IP address of the logger the **BluePiratClient** (S. 24) instance should access.*
- std::vector< **bp::LoggerInNetwork** > **getListOfLoggersInNetwork** (unsigned searchTimeOut=0)
  Returns a vector with all loggers in the network.
- void **initTransfer** ()
  Initialisation of the data transfer.
- void **disconnectLogger** ()
  Disconnects the logger that was initialised with initTransfer.
- void **initOfflineConversion** (std::string offlineArchive)
  Initialisation of the offline conversion process.
- void **keepLoggerAlive** (std::string ip="")
  Call this to keep logger alive.
- void **stopKeepLoggerAlive** (std::string ip="")
  Called to stop sending keep alive ping to the logger specified via the passed ip.
- std::vector< **Channel** > **getChannelVector** ()
  Call this function after calling the init function to receive a vector of channels.
- **LoggerConfiguration readConfigurationFromLogger** ()
  Returns the current logger configuration.
- void **writeConfigurationToLogger** (**LoggerConfiguration** lc)
  Sets the logger configuration.
- std::vector< **TimeZone** > **getTimeZones** ()

Returns all available time zones.

- void **setClientConfiguration** (**ClientConfiguration** &cc)

  Sets the client configurations.
- **EventContainer getEventContainer** ()

  Returns the list of events in a proprietary format.
- std::vector< ConvertedFileInfo > **downloadAndConvertData** (**EventContainer** container)

  *Downloads and converts the data of the selected events in the passed **EventContainer** (S. 62).*
- std::vector< ConvertedFileInfo > **convertOfflineData** (**EventContainer** container)

  *Converts the data of the selected events in the passed **EventContainer** (S. 62).*
- std::vector< ConvertedFileInfo > **downloadAndConvertData** (**TimeSpanContainer** container)

  *Downloads and converts the data of the time spans in the passed **TimeSpanContainer** (S. 117).*
- std::vector< ConvertedFileInfo > **convertOfflineData** (**TimeSpanContainer** container)

  *Converts the data of the time spans in the passed **TimeSpanContainer** (S. 117).*
- void **downloadData** (**EventContainer** container, std::string offlineDataTarget, bool sortedDownload=true)

  *Downloads the data of the selected events in the passed **EventContainer** (S. 62).*
- void **downloadData** (**TimeSpanContainer** container, std::string offlineDataTarget, bool sortedDownload=true)

  *Downloads the data of the time spans in the passed **TimeSpanContainer** (S. 117).*
- void **deleteAllFilesOnLogger** ()

  Deletes all recorded trace files on the data logger.
- std::string **setLoggerTime** (time_t time)

  Set the internal time of the data logger to the passed time.
- std::string **setLoggerTimeToSystemTime** ()

  Sets the internal time of the data logger to the time of the connected system.
- std::string **getDataLoggerName** ()

  Returns the data logger name stored on the logger.
- void **unprotectAllFilesOnLogger** ()

  Removes the protection of marked trace data.
- bool **deleteData** (**EventContainer** container, bool traceData=0, bool cameraData=0)

  *Deletes the data associated with the selected events in the **EventContainer** (S. 62).*
- bool **deleteData** (**TimeSpanContainer** container, bool traceData=0, bool cameraData=0)

  Deletes all data within the passed time spans.
- **LoggerStatus detectLogger** ()

  Searchs for a blue PiraT logger in the PCs network and returns the status.
- void **appendEvent** (std::string message)

  This function appends the passed message text to the event overview.
- time_t **getCurrentLoggerTime** ()

  Returns the current loggertime in seconds since 01.01.1970 UTC.
- void **downloadLogFiles** (std::string directoryPath)

  Downloads several log files from tue logger to the specified directory.
- void **setDebugLevel** (unsigned int level)

  Set the debug level for more detailed debug output, 1 = default, 2 = extended.
- void **addBluePiratClientListener** (**BluePiratClientListener** *listener)

  Adds a listener to the library.

- void **addTransferApplicationListener** (**TransferApplicationListener** ∗listener)

  Adds a listener to the library - this function is deprecaded. Use addBluePiratClientListener instead!
- std::string **getLibVersion** ()

  Returns the current client library version.
- void **connectLogger** ()

  Creates a connection to the data logger with the passed IP passed to the constructor.
- int **setMarker** ()

  Appends a marker entry to the Events file and displays it to the user (i.e., trigger LED at the front panel flashes, remote control beeps and displays message)

9.1.1 Ausführliche Beschreibung

Interface Class for the blue PiraT Client library.

This class provides the interface functions to allow other applications the usage of the blue PiraT library.

Example:

```
//
      ******************************************************************************
//
      ******************************************************************************
// $Id$
//
      ******************************************************************************
//
      ******************************************************************************

#include <iostream>
#include <conio.h>
#include <direct.h>

#include "BluePiratClient.hh"
#include "ClientConfiguration.hh"
#include "BluePiratClientException.hh"
#include "Channel.hh"

using namespace std;
using namespace bp;


//=====================================================================
// Dies ist ein Minimalbeispiel zur Verwendung der blue PiraT Client Library.
// Es werden alle Tracedaten seit dem letzten Aufstarten des angeschlossenen
// Datenloggers in ein Verzeichnis "OfflineData" im temporären Verzeichnis
       c:\Temp\bpc-test
// herunter geladen.
//=====================================================================
void main()
{
    try
    {
        _mkdir("C:/Temp/bpc-test/");

        // Konstruktor
        BluePiratClient bpc("C:/Temp/bpc-test/");

        /* Soll ein bestimmter Logger ausgelesen werden, muss die Funktion
       BluePiratClient::setLoggerIP()
           aufgerufen werden. Eine Liste mit allen im Netzwerk befindlichen
       Loggern liefert
```

```
        BluePiratClient::getListOfLoggersInNetwork()
        Wird keine IP-Adresse gesetzt, versucht sich die Library mit dem
     Standard-Logger
        zu verbinden. */

      // bpc.setLoggerIP(ip);

      // Initialisierung der Datenübertragung
      bpc.initTransfer();

      EventContainer events = bpc.getEventContainer();
      int i;
      bool lastStartupFound = false;
      for (i = events.getNumEntries() - 1; i >= 0; i--)
      {
          // selektiert nur den letzten Startup im Container zum download
          if (!lastStartupFound && events.getEntry(i).getEventType() ==
    STARTUP)
          {
              events.getEntry(i).setTransferFlag(true);
              lastStartupFound = true;
          }
          else
              events.getEntry(i).setTransferFlag(false);
      }

      // Konfiguration der Konvertierung
      ClientConfiguration config;
      config.setChannelToFormat(Channel::CAN, 0, CANOE);
      config.setChannelToFormat(Channel::CAN, 1, CANOE);
      config.setChannelToFormat(Channel::MOST_CTRL, 0, OP2);

      config.setTargetDirectory("C:/Temp/bpc-test/");
      config.setNameOfTester("Tester");
      config.useLongFileNameFormat(true);

      // Übergeben der Konfiguratio an die Library
      bpc.setClientConfiguration(config);

      // Herunterladen und konvertieren der Tracedaten
      bpc.downloadAndConvertData(events);
  }
  catch(BluePiratClientException exception)
  {
      cout << "BPC ERROR: " << exception.getMessage() << endl;
      cout << "Druecken Sie 'q' zum Beenden" << endl;
      char in = getch();
      while (in != 'q' && in != 'Q')//cin >> ret;
          in = getch();
      cin.clear();

      return;
  }
}
```

9.1.2 Beschreibung der Konstruktoren und Destruktoren

9.1.2.1 BluePiratClient (const char * *tempDirectory,* const char * *loggerIP =* "")

Constructor, needs the path to a temporary directory as parameter.

The constructor needs the path to a valid temp directory. If an invalid path is given, the call of the **initTransfer()** (S. 28) function will fail and the entire data transfer process will be aborted with an error message notified by a thrown exception.

Benutzt BPC_LIB_VERSION, BluePiratClientException::COMPILER_ERROR, TransferApplicationException::getMessage(), BluePiratClientException::INVALID_VALUE, BluePiratClientException::LIB_VERSION_ERROR und BluePiratClientException::TRANSFER_ERROR.

9.1.2.2 ~BluePiratClient ( )

Destructor.

9.1.3 Dokumentation der Elementfunktionen

9.1.3.1 void setLoggerIP (std::string *ip*)

Sets the IP address of the logger the **BluePiratClient** (S. 24) instance should access.

This function overwrites the parameter *loggerIP* passed to the constructor- All further function calls will communicate with the logger owning this ip.

9.1.3.2 std::vector< bp::LoggerInNetwork > getListOfLoggersInNetwork (unsigned *searchTimeOut* = 0)

Returns a vector with all loggers in the network.

During the passed time period, the network is periodically scanned for loggers. All found loggers a stored in the returned vector and simultaneously reported via the **LoggerDetectorListener** (S. 90) functions.

Siehe auch

LoggerDetectorListener (S. 90)

9.1.3.3 void initTransfer ( )

Initialisation of the data transfer.

Initialises the data transfer process by connecting to the logger, reading the data logger configuration and initialising the internal **EventContainer** (S. 62) (Startup-, Marker-, Shutdown--Information), which can be received afterwards from the getEventContainer function. This function has to be called first of all if you want to call one of the functions **downloadAndConvertData()** (S. 31) or **downloadData()** (S. 32).

When calling this function, the data logger must be connected and active.

Benutzt TransferApplicationException::getMessage(), BluePiratClientException::INVALID_VALUE, BluePiratClientException::LICENSE_ERROR und BluePiratClientException::TRANSFER_ERROR.

9.1.3.4 void disconnectLogger ( )

Disconnects the logger that was initialised with initTransfer.

The function **initTransfer()** (S. 28) initializes the data transfer process either for the default logger or for the logger with the IP set by **setLoggerIP()** (S. 28). Before initializing a data transfer for another logger in the network, this function has to be called followed by the **setLoggerIP()** (S. 28) and **initTransfer()** (S. 28) function.

9.1.3.5 void initOfflineConversion (std::string *offlineArchivePath*)

Initialisation of the offline conversion process.

Initialises the overall process for offline conversion. The internal **EventContainer** (S. 62) is initialised and can be received afterwards from the function getEventContainer. This function has to be called first of all if you want to call the function **convertOfflineData()** (S. 31). The passed *offlineArchivePath* must lead to a directory or ZIP file that contains data from a previous trace data download.

Siehe auch

downloadData() (S. 32)

Benutzt TransferApplicationException::getMessage(), BluePiratClientException::INVALID_VALUE, BluePiratClientException::LICENSE_ERROR und BluePiratClientException::TRANSFER_ERROR.

9.1.3.6 void keepLoggerAlive (std::string *ip* = "")

Call this to keep logger alive.

Since firmware 5.0.0 the data logger can be kept awake by calling this function. The **BluePiratClient** (S. 24) instance sends UDP messages to the logger to avoid a shutdown. This is generally required for blue PiraT E logger, since they don't consider the ethernet link status anymore to decide about a shutdown. The logger would shut down during a data download if there is no further bus traffic. Once called, the UDP message are sent during the entire lifespan of the **BluePiratClient** (S. 24) instance in a separate thread. For all logger types beside the blue PiraT E the network timeout setting of the logger configuration is crucial for the logger's shutdown timeout. Using this function will also avoid a shutdown of those logger types after expired network timeout.

Benutzt TransferApplicationException::getMessage() und BluePiratClientException::TRANSFER_ERROR.

9.1.3.7 void stopKeepLoggerAlive (std::string *ip* = "")

Called to stop sending keep alive ping to the logger specified via the passed *ip*.

By default, the passed parameter is empty. This means to stop all threads that send periodically pings to one logger in the network to keep it alive. Passing a valid IP address will only stop the thread for this logger. The destructor of **BluePiratClient** (S. 24) will stop all running threads.

9.1.3.8 std::vector< Channel > getChannelVector ( )

Call this function after calling the init function to receive a vector of channels.

This function returns a vector of all existing channels on the logger/offline data set.

9.1.3.9 LoggerConfiguration readConfigurationFromLogger ( )

Returns the current logger configuration.

To change a logger's configuration it is strongly recommended to modify an instance of **LoggerConfiguration** (S. 82) that was received from the **readConfigurationFromLogger()** (S. 30) function and pass the modified object back via the **writeConfigurationToLogger()** (S. 30) function. Otherwise, using an own instance of **LoggerConfiguration** (S. 82), all configuration settings, except the modified ones, will be overwritten with the default values.

Benutzt bp::DEVICE_CAPABILITIES, bp::FIRMWARE_CAPABILITIES, TransferApplicationException::getMessage(), bp::LICENSES, LoggerConfiguration::reset() und BluePiratClientException::TRANSFER_ERROR.

9.1.3.10 void writeConfigurationToLogger (LoggerConfiguration *lc*)

Sets the logger configuration.

Updates the logger's configuration with the passed configuration

Siehe auch

readConfigurationFromLogger (S. 30)

Benutzt bp::CONFIG_WRITE_TO_LOGGER, BluePiratClientException::CONFIGURATION_ERROR, LoggerConfiguration::d_canFilterPropertiesVector, LoggerConfiguration::d_canPropertiesVector, LoggerConfiguration::d_cascadingProperties, LoggerConfiguration::d_dataStorageProperties, LoggerConfiguration::d_flexrayGeneralProperties, LoggerConfiguration::d_flexrayPropertiesVector, LoggerConfiguration::d_generalProperties, LoggerConfiguration::d_linPropertiesVector, LoggerConfiguration::d_markerProperties, LoggerConfiguration::d_markerProtectionProperties, LoggerConfiguration::d_mostFilterProperties, LoggerConfiguration::d_mostProperties, LoggerConfiguration::d_networkConfigProperties, LoggerConfiguration::d_rcVoiceProperties, LoggerConfiguration::d_serialPropertiesVector, LoggerConfiguration::d_sleepProperties, bp::DEVICE_CAPABILITIES, bp::FIRMWARE_CAPABILITIES, NetworkConfigProperties::getDHCPMode(), NetworkConfigProperties::getIP(), MarkerProperties::getMarkerMessageData(), MarkerProperties::getMarkerMessageMask(), TransferApplicationException::getMessage(), NetworkConfigProperties::getNetmask(), BluePiratClientException::LICENSE_ERROR, bp::LICENSES und BluePiratClientException::TRANSFER_ERROR.

9.1.3.11 std::vector< TimeZone > getTimeZones ( )

Returns all available time zones.

The returned vector includes all time zones known by the system. To configure the logger with a new time zone, select one from the vector and pass its index to the **GeneralProperties::setTimezoneIndex()** (S. 75).

9.1.3.12 void setClientConfiguration (ClientConfiguration & *cc*)

Sets the client configurations.

9.1.3.13 EventContainer getEventContainer ( )

Returns the list of events in a proprietary format.

Returns the list of events as instance of the class **EventContainer** (S. 62). Before calling this function, the returned **EventContainer** (S. 62) has to be initialised by one of the functions initTransfer or initOfflineConversion. One event is stored as **EventContainerEntry** (S. 67) in the returned object. An entry contains the information about the event type (see EventType in **TypeDefs.hh** (S. 156)), the event time stamp and whether the event is selected for transfer or not (transfer status). The returned container can be modified by setting the transfer status of the included events. Passing the modified container to the function tranferData() or **convertOfflineData()** (S. 31) will download and/or convert all data that is associated with the selected events. Selecting a SHUTDOWN event has no effect. Those events are only for limitation of the resulting time span of a STARTUP event.

Siehe auch

transferData(EventContainer container)
EventContainerEntry (S. 67), **EventContainer** (S. 62)

9.1.3.14 std::vector< ConvertedFileInfo > downloadAndConvertData (EventContainer *container*)

Downloads and converts the data of the selected events in the passed **EventContainer** (S. 62).

Passing the modified **EventContainer** (S. 62) to this function will download and convert all data that is associated with the selected events.

The overall process must be first initialised by the **initTransfer()** (S. 28) function.

Siehe auch

getEventContainer() (S. 31)
EventContainerEntry (S. 67), **EventContainer** (S. 62)

Benutzt TransferApplicationException::getMessage() und BluePiratClientException::TRANSFER_ERROR.

9.1.3.15 std::vector< ConvertedFileInfo > convertOfflineData (EventContainer *container*)

Converts the data of the selected events in the passed **EventContainer** (S. 62).

This function converts the raw trace data of a prior data download that is associated with the selected events in the passed **EventContainer** (S. 62).

The overall process must be first initialised by the **initOfflineConversion()** (S. 29) function.

Siehe auch

downloadData() (S. 32)
getEventContainer() (S. 31)
EventContainerEntry (S. 67), **EventContainer** (S. 62)

Benutzt TransferApplicationException::getMessage() und BluePiratClientException::TRANSFER_ERROR.

9.1.3.16 std::vector< ConvertedFileInfo > downloadAndConvertData (TimeSpanContainer *container*)

Downloads and converts the data of the time spans in the passed **TimeSpanContainer** (S. 117).

This function downloads and converts all available data within the passed **TimeSpanContainer** (S. 117).

The overall process must be first initialised by the **initTransfer()** (S. 28) function.

Siehe auch

TimeSpanContainer (S. 117), **TimeSpan** (S. 115)

Benutzt TransferApplicationException::getMessage() und BluePiratClientException::TRANSFER_ERROR.

9.1.3.17 std::vector< ConvertedFileInfo > convertOfflineData (TimeSpanContainer *container*)

Converts the data of the time spans in the passed **TimeSpanContainer** (S. 117).

This function converts the raw trace data of a prior data download that lies within the passed **TimeSpanContainer** (S. 117).

The overall process must be first initialised by the **initOfflineConversion()** (S. 29) function.

Siehe auch

downloadData() (S. 32)
TimeSpanContainer (S. 117), **TimeSpan** (S. 115)

Benutzt TransferApplicationException::getMessage() und BluePiratClientException::TRANSFER_ERROR.

9.1.3.18 void downloadData (EventContainer *container*, std::string *offlineDataTarget*, bool *sortedDownload* = `true`)

Downloads the data of the selected events in the passed **EventContainer** (S. 62).

This function downloads the raw trace data associated with the selected events in the passed **EventContainer** (S. 62) to a local destination specified by *offlineDataTarget*. This parameter can either be a directory or the name of a ZIP file. If the directory already exists, it must be empty. If not, it will be created automatically. The specified ZIP file must not be existing, otherwise an exception is thrown. When downloading to a ZIP archive, the library has to regard the max file size of such file systems. Therefore the ZIP will be devided at a max file size of 3,8 GB and will extend

the file name /*offlineDataTarget*) by an appending index ("_#1, _#2, etc). The calling application will be informed by the function **BluePiratClientListener::onOfflineZIPRename** (S. 128). Please se documentation of this function. The downloaded data can then be used for a subsequent conversion.

The overall process must be first initialised by the **initTransfer()** (S. 28) function.

Siehe auch

convertOfflineData() (S. 31)
getEventContainer() (S. 31)
EventContainerEntry (S. 67), **EventContainer** (S. 62)

Benutzt TransferApplicationException::getMessage() und BluePiratClientException::TRANSFER_ERROR.

9.1.3.19 void downloadData (TimeSpanContainer *container*, std::string *offlineDataTarget*, bool *sortedDownload* = `true`)

Downloads the data of the time spans in the passed **TimeSpanContainer** (S. 117).

This function downloads all available raw data within the passed **TimeSpanContainer** (S. 117) to a local destination specified by *offlineDataTarget*. This parameter can either be a directory or the name of a ZIP file. If the directory already exists, it must be empty. If not, it will be created automatically. The specified ZIP file must not be existing, otherwise an exception is thrown.

The downoaded data can then be used for a subsequent conversion.

The overall process must be first initialised by the **initTransfer()** (S. 28) function.

Siehe auch

convertOfflineData() (S. 31)
TimeSpanContainer (S. 117), **TimeSpan** (S. 115)

Benutzt TransferApplicationException::getMessage() und BluePiratClientException::TRANSFER_ERROR.

9.1.3.20 void deleteAllFilesOnLogger ( )

Deletes all recorded trace files on the data logger.

Benutzt TransferApplicationException::getMessage() und BluePiratClientException::TRANSFER_ERROR.

9.1.3.21 std::string setLoggerTime (time_t *time*)

Set the internal time of the data logger to the passed *time*.

The passed *time* is from type time_t defined in <time.h>. It represents the number of seconds elapsed since midnight on January 1, 1970.

Rückgabe

Confirmation string for setting time.

Benutzt TransferApplicationException::getMessage() und BluePiratClientException::TRANSFER_ERROR.

9.1.3.22 std::string setLoggerTimeToSystemTime ( )

Sets the internal time of the data logger to the time of the connected system.

Sets the time to the PC's time.

Rückgabe

Confirmation string for setting time.

Benutzt TransferApplicationException::getMessage() und BluePiratClientException::TRANSFER_ERROR.

9.1.3.23 std::string getDataLoggerName ( )

Returns the data logger name stored on the logger.

Benutzt TransferApplicationException::getMessage() und BluePiratClientException::TRANSFER_ERROR.

9.1.3.24 void unprotectAllFilesOnLogger ( )

Removes the protection of marked trace data.

The data logger can be configured such that data that is marked by a marker event is protected against overwriting in case of activated ring buffer mode and full data logger hard drive. This function unprotects all data on the logger that was traced in the past.

Benutzt TransferApplicationException::getMessage() und BluePiratClientException::TRANSFER_ERROR.

9.1.3.25 bool deleteData (EventContainer *container*, bool *traceData* = 0, bool *cameraData* = 0)

Deletes the data associated with the selected events in the **EventContainer** (S. 62).

To select the data that should be deleted, the transfer flag of the event containers' entries is used. Select those events whose data should be deleted. Because of the different storing locations for camera and trace data you can specifiy which type of data should be deleted by setting the appropriate falgs *traceData* and *cameraData*.

The **initTransfer()** (S. 28) function has to be called before.

Benutzt TransferApplicationException::getMessage() und BluePiratClientException::TRANSFER_ERROR.

9.1.3.26 bool deleteData (TimeSpanContainer *container*, bool *traceData* = 0, bool *cameraData* = 0)

Deletes all data within the passed time spans.

Specifiy those time spans in the **TimeSpanContainer** (S. 117) *container* for that the data should be deleted. Because of the different storing locations for camera and trace data you can specifiy which type of data should be deleted by setting the appropriate flags *traceData* and *cameraData*.

The **initTransfer()** (S. 28) function has to be called before.

Benutzt TransferApplicationException::getMessage() und BluePiratClientException::TRANSFER_ERROR.

9.1.3.27 LoggerStatus detectLogger ( )

Searchs for a blue PiraT logger in the PCs network and returns the status.

Benutzt TransferApplicationException::getMessage(), bp::LOGGER_FOUND, bp::LOGGER_NOT_FOUND und BluePiratClientException::TRANSFER_ERROR.

9.1.3.28 void appendEvent (std::string *message*)

This function appends the passed message text to the event overview.

The set entry can be treaded like a marker entry. Its EventType is INFO.

Benutzt TransferApplicationException::getMessage() und BluePiratClientException::TRANSFER_ERROR.

9.1.3.29 time_t getCurrentLoggerTime ( )

Returns the current loggertime in seconds since 01.01.1970 UTC.

Benutzt TransferApplicationException::getMessage() und BluePiratClientException::TRANSFER_ERROR.

9.1.3.30 void downloadLogFiles (std::string *directoryPath*)

Downloads several log files from tue logger to the specified *directory*.

Benutzt TransferApplicationException::getMessage() und BluePiratClientException::TRANSFER_ERROR.

9.1.3.31 void setDebugLevel (unsigned int *level*)

Set the debug level for more detailed debug output, 1 = default, 2 = extended.

9.1.3.32 void addBluePiratClientListener (BluePiratClientListener * *listener*)

Adds a listener to the library.

9.1.3.33 void addTransferApplicationListener (TransferApplicationListener * *listener*)

Adds a listener to the library - this function is deprecaded. Use addBluePiratClientListener instead!

The library forwards all kind of exceptions that don't abort the transfer and conversions process to its listener.

9.1.3.34 std::string getLibVersion ( )

Returns the current client library version.

Benutzt BPC_LIB_VERSION.

9.1.3.35 void connectLogger ( )

Creates a connection to the data logger with the passed IP passed to the constructor.

Benutzt TransferApplicationException::getMessage() und BluePiratClientException::TRANSFER_ERROR.

9.1.3.36 int setMarker ( )

Appends a marker entry to the Events file and displays it to the user (i.e., trigger LED at the front panel flashes, remote control beeps and displays message)

The execution of this function may take a few seconds.

Benutzt TransferApplicationException::getMessage() und BluePiratClientException::TRANSFER_ERROR.

9.2 BluePiratClientException Klassenreferenz

Class for blue PiraT Client library errors.

Öffentliche Typen

- enum **ExceptionType** {
  TRANSFER_ERROR, **INVALID_VALUE**, **CONFIGURATION_ERROR**, **LICENSE_ERROR**,
  COMPILER_ERROR, **LIB_VERSION_ERROR** }
  List of possible exception types.

Öffentliche Methoden

- **BluePiratClientException** (std::string message, **ExceptionType** type)
  Creates a new exception with a specified message.
- std::string **getMessage** ()

Returns the error message of the exception.

- **ExceptionType getType** ()

  Returns the type of the exception.

9.2.1 Ausführliche Beschreibung

Class for blue PiraT Client library errors.

The exception handling of all library functions

9.2.2 Dokumentation der Aufzählungstypen

9.2.2.1 enum ExceptionType

List of possible exception types.

Aufzählungswerte:

TRANSFER_ERROR
INVALID_VALUE
CONFIGURATION_ERROR
LICENSE_ERROR
COMPILER_ERROR
LIB_VERSION_ERROR

9.2.3 Beschreibung der Konstruktoren und Destruktoren

9.2.3.1 BluePiratClientException (std::string *message,* ExceptionType *type*) [inline]

Creates a new exception with a specified message.

The last Windows system error is inquired by GetLastError() when creating the Exception instance.

9.2.4 Dokumentation der Elementfunktionen

9.2.4.1 std::string getMessage ( ) [inline]

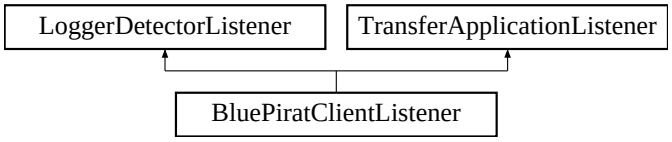
Returns the error message of the exception.

9.2.4.2 ExceptionType getType ( ) [inline]

Returns the type of the exception.

9.3 BluePiratClientListener Klassenreferenz

Klassendiagramm für BluePiratClientListener:



9.4 CANChannelClockFrequency Strukturreferenz

Type definition for the Configuration Tool.

Öffentliche Attribute

- std::string **channelName**
- int **clockFrequency**

9.4.1 Ausführliche Beschreibung

Type definition for the Configuration Tool.

9.4.2 Dokumentation der Datenelemente

9.4.2.1 std::string channelName

9.4.2.2 int clockFrequency

9.5 CanFilterProperties Klassenreferenz

Property class to store the filtered CAN IDs.

Öffentliche Methoden

- **CanFilterProperties** ()
  Constructor.
- void **setActive** (bool status)
  Activates/deactivates the CAN filter.
- void **setCanIDs** (std::vector< uint32_t > canIDs)
  Sets the CAN IDs that should be recorded.
- void **setChannel** (int filterChannel)
  Sets the CAN channel for that the CAN IDs should be filtered.

- bool **isActive** ()

  Returns the active status.
- std::vector< uint32_t > **getCanIDs** ()

  Returns the CAN filter IDs.
- int **getChannel** ()

  Returns the filter's CAN channel.
- virtual bool **setPropertiesToDefaultConfig** ()

9.5.1 Ausführliche Beschreibung

Property class to store the filtered CAN IDs.

The data logger can be configured to record only CAN messages with certain CAN IDs. The IDs to filter have to be stored separated for each CAN channel in one instance of this class. The CAN filters can be activated and deactivated by the function setActive.

9.5.2 Beschreibung der Konstruktoren und Destruktoren

9.5.2.1 CanFilterProperties ( )

Constructor.

Constructor that initializes the CAN filter properties with default values.

Benutzt CanFilterProperties::setPropertiesToDefaultConfig().

9.5.3 Dokumentation der Elementfunktionen

9.5.3.1 void setActive (bool *status*)

Activates/deactivates the CAN filter.

An active filter let the logger only record the set CAN IDs. If on CAN IDs are set, the logger won't record any CAN message at the filter's CAN channel.

Parameter

status	Filter active status.

9.5.3.2 void setCanIDs (std::vector< uint32_t > *canIDs*)

Sets the CAN IDs that should be recorded.

All CAN messages at the specified CAN channel with the CAN IDs set with this function are recorded by the data logger. Note that the filter must be activated by the setActive function.

Parameter

canIDs	Vector of CAN IDs.

9.5.3.3 void setChannel (int *filterChannel*)

Sets the CAN channel for that the CAN IDs should be filtered.

Wird benutzt von LoggerConfiguration::getCanFilterProperties().

9.5.3.4 bool isActive ( )

Returns the active status.

9.5.3.5 std::vector< uint32_t > getCanIDs ( )

Returns the CAN filter IDs.

9.5.3.6 int getChannel ( )

Returns the filter's CAN channel.

Wird benutzt von LoggerConfiguration::setCanFilterProperties().

9.5.3.7 bool setPropertiesToDefaultConfig ( ) [virtual]

Wird benutzt von CanFilterProperties::CanFilterProperties().

9.6 CanProperties Klassenreferenz

Property class to store the CAN properties of a CAN interface.

Öffentliche Typen

- enum **CanBaudrate** {
  CBAUD_33333, **CBAUD_50000**, **CBAUD_83333**, **CBAUD_100000**,
  CBAUD_125000, **CBAUD_250000**, **CBAUD_500000**, **CBAUD_1000000** }
  List of all possible CAN baud rates.

Öffentliche Methoden

- **CanProperties** ()
  Constructor.
- void **setChannel** (int id)
  Sets the channel index starting with 0.
- int **getChannel** ()
  Returns the channel's index.
- void **setName** (std::string name)
  Sets the CAN channel name.

- std::string **getName** ()

  Returns channel name.
- void **setBaudRate** (**CanBaudrate** baudRate)

  Sets the baud rate of the CAN interface.
- **CanProperties::CanBaudrate getBaudRate** ()

  Returns channel's baud rate.
- void **setTransceiverType** (bool type)

  Sets the CAN transceiver type (high/low)
- bool **getTransceiverType** ()

  Returns true for high speed transeiver type, false for low speed.
- void **setActive** (bool channelActive)

  Activates/deactivates the channel.
- bool **isActive** ()

  Returns the status of the CAN channel (active/deactive)
- void **setAcknowledge** (bool acknowledgeStatus)

  Sets the CAN channel's acknowledge status.
- bool **isAcknowledge** ()

  Returns acknowledge status.
- void **setBTRModeActive** (bool BTRModeStatus)

  Activates/deactivates the bit timing register mode.
- bool **isBTRModeActive** ()

  Returns the status of the bit timing register mode.
- void **setBTR0Value** (uint8_t BTR0)

  Sets the value of the bit timing register's byte 0.
- uint8_t **getBTR0Value** ()

  Returns the value of the bit timing register's byte 0.
- void **setBTR1Value** (uint8_t BTR1)

  Sets the value of the bit timing register's byte 1.
- uint8_t **getBTR1Value** ()

  Returns the value of the bit timing register's byte 1.
- void **setParameters** (InterfaceParameters param)

  Called to set interface parameters.
- InterfaceParameters **getParameters** ()

  Returns interface parameters.
- virtual bool **setPropertiesToDefaultConfig** ()

  Called to set Properties to default settings.
- void **readFromConfigContainer** (ConfigContainer ∗configContainer)
- void **writeToConfigContainer** (ConfigContainer ∗configContainer)

  only for internal use

9.6.1 Ausführliche Beschreibung

Property class to store the CAN properties of a CAN interface.

Each CAN interface can be activated or deactivated by **setActive()** (S. 44). There is no data recorded from inactive CAN interfaces. With **setName()** (S. 43) a name can be assigned to each CAN interface, which is later added to the filenames of the trace files for easier identification. Finally, the baudrate and type (low speed/high speed) of the CAN bus are adjustable, see **setBaudRate()** (S. 43) and **setTransceiverType()** (S. 43). Use **setAcknowledge()** (S. 44) to enable or disable the acknowledge of the CAN transceiver of type high speed. (Note: If the acknowledge is disabled, the transceiver cannot send CAN messages for the marker confirmation as described in **MarkerProperties** (S. 92). The Acknowledge of transceiver of type low speed can not be deactivated. Alternatively to specifying the baudrate, it is possible to configure the CAN bit-timings directly by specifying the chip parameters. Activate this mode by the **setBTRModeActive()** (S. 44) functions and set the chip parameters with **setBTR0Value()** (S. 44) and **setBTR1Value()** (S. 45). For more information about that see the blue PiraT User's Manual, that is provided with the blue PiraT Client Software packet.

9.6.2 Dokumentation der Aufzählungstypen

9.6.2.1 enum CanBaudrate

List of all possible CAN baud rates.

Aufzählungswerte:

CBAUD_33333
CBAUD_50000
CBAUD_83333
CBAUD_100000
CBAUD_125000
CBAUD_250000
CBAUD_500000
CBAUD_1000000

9.6.3 Beschreibung der Konstruktoren und Destruktoren

9.6.3.1 CanProperties ( )

Constructor.

Constructor that initializes the CAN parameters with their default values.

Benutzt CanProperties::setPropertiesToDefaultConfig().

9.6.4 Dokumentation der Elementfunktionen

9.6.4.1 void setChannel (int *id*)

Sets the channel index starting with 0.

If the transceiver type is low speed and the channel index is set to greater than 1, the transceiver type is automatically set to high speed.

Wird benutzt von LoggerConfiguration::getCanProperties().

9.6.4.2 int getChannel ( )

Returns the channel's index.

Wird benutzt von LoggerConfiguration::setCanProperties().

9.6.4.3 void setName (std::string *name*)

Sets the CAN channel name.

The channel's name appears in the converted trace file names.

9.6.4.4 std::string getName ( )

Returns channel name.

Returns name property of the CAN bus.

9.6.4.5 void setBaudRate (CanBaudrate *baudRate*)

Sets the baud rate of the CAN interface.

Benutzt CanProperties::CBAUD_100000, CanProperties::CBAUD_1000000, CanProperties::CBAUD_125000, CanProperties::CBAUD_250000, CanProperties::CBAUD_33333, CanProperties::CBAUD_50000, CanProperties::CBAUD_500000, CanProperties::CBAUD_83333 und BluePiratClientException::INVALID_VALUE.

9.6.4.6 CanProperties::CanBaudrate getBaudRate ( )

Returns channel's baud rate.

Benutzt CanProperties::CBAUD_100000, CanProperties::CBAUD_1000000, CanProperties::CBAUD_125000, CanProperties::CBAUD_250000, CanProperties::CBAUD_33333, CanProperties::CBAUD_50000, CanProperties::CBAUD_500000 und CanProperties::CBAUD_83333.

9.6.4.7 void setTransceiverType (bool *type*)

Sets the CAN transceiver type (high/low)

Passing 1/true to this function sets the transceiver type to high speed, 0/false to low speed. In case of low speed, the CAN acknowledge status is automatically set to 'active'.

Benutzt BluePiratClientException::INVALID_VALUE.

9.6.4.8 bool getTransceiverType ( )

Returns true for high speed transeiver type, false for low speed.

9.6.4.9 void setActive (bool *channelActive*)

Activates/deactivates the channel.

Passing true to this function activates the CAN channel, false will deactivate it. Deactive channels are not recorded by the data logger

9.6.4.10 bool isActive ( )

Returns the status of the CAN channel (active/deactive)

9.6.4.11 void setAcknowledge (bool *acknowledgeStatus*)

Sets the CAN channel's acknowledge status.

Because of the hardware design of the blue PiraT, it is not possible to disable sending the acknowledge bit for low-speed channels.

9.6.4.12 bool isAcknowledge ( )

Returns acknowledge status.

9.6.4.13 void setBTRModeActive (bool *BTRModeStatus*)

Activates/deactivates the bit timing register mode.

Alternatively to specifying the baudrate, it is possible to configure the CAN bit-timings directly by specifying the chip parameters. Use the functions **setBTR0Value()** (S. 44) and **setBTR1Value()** (S. 45) to set the chip parameters.

9.6.4.14 bool isBTRModeActive ( )

Returns the status of the bit timing register mode.

9.6.4.15 void setBTR0Value (uint8_t *BTR0*)

Sets the value of the bit timing register's byte 0.

For more details see the blue PiraT User's Manual, provided with the blue PiratClient Software packet

9.6.4.16 uint8_t getBTR0Value ( )

Returns the value of the bit timing register's byte 0.

Called to get bit timing register least significant nibble (0) value.

9.6.4.17 void setBTR1Value (uint8_t *BTR1*)

Sets the value of the bit timing register's byte 1.

For more details see the blue PiraT User's Manual, provided with the blue PiratClient Software packet

9.6.4.18 uint8_t getBTR1Value ( )

Returns the value of the bit timing register's byte 1.

9.6.4.19 void setParameters (InterfaceParameters *param*)

Called to set interface parameters.

9.6.4.20 InterfaceParameters getParameters ( )

Returns interface parameters.

9.6.4.21 bool setPropertiesToDefaultConfig ( ) [virtual]

Called to set Properties to default settings.

Wird benutzt von CanProperties::CanProperties().

9.6.4.22 void readFromConfigContainer (ConfigContainer * *configContainer*)

COMPLETE_DOC only for internal use

9.6.4.23 void writeToConfigContainer (ConfigContainer * *configContainer*)

only for internal use

Benutzt bp::CAN_ACKNOWLEDGE_MODE, bp::CAN_BIT_TIMING_MODE, bp::FIRMWARE_-CAPABILITIES und bp::INTERFACE_NAME_MAPPING_FEATURE.

9.7 CascadingProperties Klassenreferenz

This class stores the properties for the logger's cascading feature.

Öffentliche Typen

- enum **CascadingMode** { **CASCADING_OFF**, **CASCADING_MASTER**, **CASCADING_SLAVE** }

  List of cascading modes.

Öffentliche Methoden

- **CascadingProperties** ()

  Constructor.
- void **setCascadingMode** (**CascadingMode** cascadingMode)

  Activates/Deactivates the cascading mode.
- **CascadingMode getCascadingMode** ()

  Returns the current cascading mode.
- void **setCANChannelOffset** (uint8_t canChannelOffset)

  Sets an offset value for the CAN channels.
- void **setSerialPortOffset** (uint8_t serialPortOffset)

  Sets an offset value for the serial ports.
- void **setLINChannelOffset** (uint8_t linChannelOffset)

  Sets an offset value for the LIN channels.
- void **setCameraPortOffset** (uint8_t cameraPortOffset)

  Sets an offset value for the camera ports.
- void **setFlexRayChannelOffset** (uint8_t flexrayChannelOffset)

  Sets an offset value for the Flexray channels.
- void **setEthernetPortOffset** (uint8_t ethernetPortOffset)

  Sets an offset value for the ethernet ports.
- uint8_t **getCANChannelOffset** ()

  Returns the current offset of the CAN channels.
- uint8_t **getSerialPortOffset** ()

  Returns the current offset of the serial ports.
- uint8_t **getLINChannelOffset** ()

  Returns the current offset of the LIN channels.
- uint8_t **getCameraPortOffset** ()

  Returns the current offset of the Camera ports.
- uint8_t **getFlexRayChannelOffset** ()

  Returns the current offset of the FlexRay channels.
- uint8_t **getEthernetIPortOffset** ()

  Returns the current offset of the ethernet ports.
- virtual bool **setPropertiesToDefaultConfig** ()

9.7.1 Ausführliche Beschreibung

This class stores the properties for the logger's cascading feature.

The blue PiraT data logger supports the trace recording with two cascaded loggers. Loggers that should work in that mode have to be configured via this property class. The cascading feature is firmware dependend. Older firmware packets eventually doesn't support this feature. The cascading settings allow to deactivate cascading (normal case for single loggers) or to designate the data logger as the master or the slave.

Important: If the cascading mode has just been activated, the synchronization is activated only at the next startup.

To allow different channel numbers of the master and the slave, it is possible to specify an offset for those channels. The channel numbers of the CAN-channels and the serial channels are corrected by this offset.

Warning: Please make sure that the data logger name of the master and the slave are different (in the **GeneralProperties** (S. 74)). Otherwise equal file names might be created for the master and slave trace files, which could cause an overwriting of the trace files during conversion.

9.7.2 Dokumentation der Aufzählungstypen

9.7.2.1 enum CascadingMode

List of cascading modes.

Aufzählungswerte:

CASCADING_OFF

CASCADING_MASTER

CASCADING_SLAVE

9.7.3 Beschreibung der Konstruktoren und Destruktoren

9.7.3.1 CascadingProperties ( )

Constructor.

Benutzt CascadingProperties::setPropertiesToDefaultConfig().

9.7.4 Dokumentation der Elementfunktionen

9.7.4.1 void setCascadingMode (CascadingProperties::CascadingMode *cascadingMode*)

Activates/Deactivates the cascading mode.

9.7.4.2 CascadingProperties::CascadingMode getCascadingMode ( )

Returns the current cascading mode.

9.7.4.3 void setCANChannelOffset (uint8_t *canChannelOffset*)

Sets an offset value for the CAN channels.

To be able to distiguish the CAN channels of two cascaded loggers it might be usefull to set an offset for the CAN channels of one logger. If the traces of both loggers are merged, the channels of the first logger (master) can have the channel IDs #0 and #1, the channels of the second logger (slave) have the IDs #2 and #3.

9.7.4.4 void setSerialPortOffset (uint8_t *serialPortOffset*)

Sets an offset value for the serial ports.

To be able to distiguish the serial ports of two cascaded loggers it might be usefull to set an offset for the ports of one logger. If the traces of both loggers are merged, the ports of the first logger (master) can have the IDs #0 and #1, the ports of the second logger (slave) have the IDs #2 and #3.

9.7.4.5 void setLINChannelOffset (uint8_t *linChannelOffset*)

Sets an offset value for the LIN channels.

9.7.4.6 void setCameraPortOffset (uint8_t *cameraPortOffset*)

Sets an offset value for the camera ports.

9.7.4.7 void setFlexRayChannelOffset (uint8_t *flexrayChannelOffset*)

Sets an offset value for the Flexray channels.

9.7.4.8 void setEthernetPortOffset (uint8_t *ethernetPortOffset*)

Sets an offset value for the ethernet ports.

9.7.4.9 uint8_t getCANChannelOffset ( )

Returns the current offset of the CAN channels.

9.7.4.10 uint8_t getSerialPortOffset ( )

Returns the current offset of the serial ports.

9.7.4.11 uint8_t getLINChannelOffset ( )

Returns the current offset of the LIN channels.

9.7.4.12 uint8_t getCameraPortOffset ( )

Returns the current offset of the Camera ports.

9.7.4.13 uint8_t getFlexRayChannelOffset ( )

Returns the current offset of the FlexRay channels.

9.7.4.14 uint8_t getEthernetIPortOffset ( )

Returns the current offset of the ethernet ports.

9.7.4.15 bool setPropertiesToDefaultConfig ( ) [virtual]

Benutzt CascadingProperties::CASCADING_OFF.

Wird benutzt von CascadingProperties::CascadingProperties().

9.8 ChangedDBPathChannel Strukturreferenz

Öffentliche Attribute

- unsigned int **d_numCanChannel**
- bool **d_isPathChanged**

9.8.1 Dokumentation der Datenelemente

9.8.1.1 unsigned int d_numCanChannel

9.8.1.2 bool d_isPathChanged

9.9 Channel Klassenreferenz

Class to represent one channel.

Öffentliche Typen

- enum **BusType** {
  MOST_CTRL = 0x00, **MOST_ASYNC** = 0x01, **CAN** = 0x02, **SERIAL** = 0x03,
  ETHERNET = 0x04, **VIDEO** = 0x05, **FLEXRAY** = 0x06, **LIN** = 0x07,
  MOST_SYNC = 0x08, **MOST_SYNC_SCAN** = 0x09, **MOST_SYNC_MAN** = 0x0A, **MOST150_CTRL** = 0x0B,
  MOST150_PACKET = 0x0C, **MOST150_ETH** = 0x0D, **APIX_CTRL** = 0x0E, **MOST50_CTRL** = 0x0F,
  MOST50_MDP = 0x10, **MOST50_ECL** = 0x11, **MOST50_SYNC** = 0x12 }

Possible bus types.

Öffentliche Methoden

- **Channel** ()
  Empty Constructor.
- **Channel** (**BusType** type, unsigned int id, std::string name="")
  Constructor that sets the bus type and the channel id.
- ~**Channel** ()
  Destructor.
- void **setBusType** (**BusType** type)
  Sets the bus type.
- **BusType getBusType** ()
  Returns the bus type.
- void **setChannelId** (unsigned int id)
  Sets the channel id.
- unsigned int **getChannelId** ()
  Returns the channel id.
- void **setName** (std::string name)
  Sets the name of the channel.
- std::string **getName** ()
  Returns the name of the channel.
- std::string **toString** ()
  Returns the channel as string.
- bool **operator==** (**Channel** channel) const

9.9.1 Ausführliche Beschreibung

Class to represent one channel.

Each instance of **Channel** (S. 49) consists of a bus type, a channel id and a name of the channel.

9.9.2 Dokumentation der Aufzählungstypen

9.9.2.1 enum BusType

Possible bus types.

Aufzählungswerte:

MOST_CTRL

MOST_ASYNC

CAN

SERIAL

ETHERNET

VIDEO

FLEXRAY
LIN
MOST_SYNC
MOST_SYNC_SCAN
MOST_SYNC_MAN
MOST150_CTRL
MOST150_PACKET
MOST150_ETH
APIX_CTRL
MOST50_CTRL
MOST50_MDP
MOST50_ECL
MOST50_SYNC

9.9.3 Beschreibung der Konstruktoren und Destruktoren

9.9.3.1 Channel ( ) [inline]

Empty Constructor.

9.9.3.2 Channel (BusType *type,* unsigned int *id,* std::string *name* = "")

Constructor that sets the bus type and the channel id.

9.9.3.3 ~Channel ( ) [inline]

Destructor.

9.9.4 Dokumentation der Elementfunktionen

9.9.4.1 void setBusType (BusType *type*)

Sets the bus type.

9.9.4.2 BusType getBusType ( )

Returns the bus type.

9.9.4.3 void setChannelId (unsigned int *id*)

Sets the channel id.

9.9.4.4 unsigned int getChannelId ( )

Returns the channel id.

9.9.4.5 void setName (std::string *name*)

Sets the name of the channel.

9.9.4.6 std::string getName ( )

Returns the name of the channel.

9.9.4.7 std::string toString ( )

Returns the channel as string.

Wird benutzt von __declspec() und ClientConfiguration::setChannelToFormat().

9.9.4.8 bool operator== (Channel *channel*) const

9.10 ClientConfiguration Klassenreferenz

Class to configure the client library settings.

Öffentliche Methoden

- **ClientConfiguration** ()
  Empty Constructor.
- **ClientConfiguration** (const **ClientConfiguration** &cc)
  Copy Constructor.
- ~**ClientConfiguration** ()
  Destructor.
- void **operator=** (const **ClientConfiguration** &cc)
  Assign operator.
- bool **init** (**BluePiratClient** *bpc)
- void **saveSettings** (std::string filePathAndName)
  Saves the current settings to a file.
- void **loadSettings** (std::string filePathAndName)
  Loads settings from a file.
- void **setTargetDirectory** (std::string path)
  Sets the path to the target directory where the converted data is stored.
- std::string **getTargetDirectory** ()
  Returns the path to the target directory where the converted data is stored.
- void **setNameOfTester** (std::string name)
  Sets the name of the tester.

- std::string **getNameOfTester** ()

  Returns the name of the tester.
- void **useLongFileNameFormat** (bool flag)

  Sets the output file name format to use.
- bool **isLongFileNameFormat** ()

  Returns the output file name format to use.
- void **setMaxOutputFileSize** (int size)

  Sets the maximum size of the converted output files.
- int **getMaxOutputFileSize** ()

  Returns the maximum size of the converted putput files.
- void **setMidnightSplitting** (bool flag)

  Sets the midnight splitting option for file conversion.
- bool **isMidnightSplitting** ()

  Returns the midnight splitting option of the file conversion.
- void **useEventTimeInFileNames** (bool flag)

  Sets the rule how to create the output file names.
- bool **isEventTimeInFileNames** ()

  Returns the rule how to create the output file names.
- void **setChannelToFormat** (**Channel::BusType** bus, unsigned int channelId, **FormatId** fid)

  *Sets the channel identified by **Channel::BusType** (S. 50), channel ID to the passed FormatId.*
- **FormatId getFormatOfChannel** (**Channel::BusType** bus, unsigned int channelId)

  Returns the format of ChannelTypeId and a channelId.
- void **setInterruptTracingDuringDownload** (bool flag)

  Sets the option to interrupt data recording during data download.
- bool **isInterruptTracingDuringDownload** ()

  Returns wether to interrupt the data recording during data download.
- void **setMarkerTimeSpanForChannelType** (**Channel::BusType** bus, int preTime, int postTime)
- void **getMarkerTimeSpanForChannelType** (**Channel::BusType** bus, int &preTime, int &postTime)
- void **setMarkerTimeSpanForChannelType** (**Channel::BusType** bus, **LowerBoundType** lower, **UpperBoundType** upper, int preTime, int postTime, std::string infoText)

  Sets the time span around the marker for that the data should be transfered (bus specific).
- void **getMarkerTimeSpanForChannelType** (**Channel::BusType** bus, **LowerBoundType** &lower, **UpperBoundType** &upper, int &preTime, int &postTime, std::string &infoText)

  Writes time span definitions around the marker for that the data should be be transferred to the passed values (bus specific).
- void **setAlternativeLoggerName** (std::string name)

  Sets an alternative data logger name.
- std::string **getAlternativeLoggerName** ()

  Returns the alternative data logger name.
- void **useAlternativeLoggerName** (bool flag)

  Defines usage of the alternative logger name for the output file names.
- bool **isAlternativeLoggerName** ()

  Returns the usage of the alternative logger name for the output file names.
- void **useSubdirectoriesForOutputTraces** (bool flag)

  Defines usage of subdirectories for storage of the converted files.
- bool **isSubdirectoriesForOutputTraces** ()

  Returns the usage of subdirectories for storage of the converted files.

9.10.1 Ausführliche Beschreibung

Class to configure the client library settings.

All settings that consider the client part of the blue PiraT Client library are stored in one instance of this class

9.10.2 Beschreibung der Konstruktoren und Destruktoren

9.10.2.1 ClientConfiguration ( )

Empty Constructor.

Class to configure the client library settings.

All settings that consider the client part of the blue PiraT Client library are stored in one instance of this class

9.10.2.2 ClientConfiguration (const ClientConfiguration & *cc*)

Copy Constructor.

9.10.2.3 ~ClientConfiguration ( )

Destructor.

9.10.3 Dokumentation der Elementfunktionen

9.10.3.1 void operator= (const ClientConfiguration & *cc*)

Assign operator.

9.10.3.2 bool init (BluePiratClient * *bpc*)

Initialise the client configuration properties, depending on the the logger's hardware version (number and type of existing channels etc.) This can be used to allow only format settings for existing channels. Must be called after the **BluePiratClient** (S. 24) parameter was initialised

The passed **BluePiratClient** (S. 24) pointer has to be initialised by one of the functions **BluePiratClient::initTransfer** (S. 28) or **BluePiratClient::initOfflineConversion** (S. 29). Note that all format and marker time span settings that were made before with function calls of **ClientConfiguration::setChannelToFormat** (S. 57) and **ClientConfiguration::setMarkerTimeSpanForChannelType** (S. 58) will be discarded when calling this function.

9.10.3.3 void saveSettings (std::string *filePathAndName*)

Saves the current settings to a file.

Benutzt BluePiratClientException::INVALID_VALUE.

9.10.3.4 void loadSettings (std::string *filePathAndName*)

Loads settings from a file.

If no settings are loaded the library uses the default values for all settings.

Benutzt BluePiratClientException::INVALID_VALUE.

9.10.3.5 void setTargetDirectory (std::string *path*)

Sets the path to the target directory where the converted data is stored.

This directory is only required for a data conversion. It specifies the path where to store the converted data. If you only want to download offline data sets, you have to pass the target directory for those directly to the downloadData() function.

In case of an invalid passed path an exception is thrown.

Siehe auch

loadSettings() (S. 55), downloadData()

Benutzt BluePiratClientException::INVALID_VALUE.

9.10.3.6 std::string getTargetDirectory ( )

Returns the path to the target directory where the converted data is stored.

9.10.3.7 void setNameOfTester (std::string *name*)

Sets the name of the tester.

The passed name is included in the output file names.

Siehe auch

loadSettings() (S. 55)

Benutzt BluePiratClientException::INVALID_VALUE.

9.10.3.8 std::string getNameOfTester ( )

Returns the name of the tester.

9.10.3.9 void useLongFileNameFormat (bool *flag*)

Sets the output file name format to use.

If *flag* is true, the time stamps in the output file names are separated by ':' and '.'

e.g. [2005-10-14]_12.02.34 instead of *20051014_120234*

Siehe auch

loadSettings() (S. 55)

9.10.3.10 bool isLongFileNameFormat ( )

Returns the output file name format to use.

Returns the output file name format to use.

Siehe auch

useLongFileNameFormat (S. 55)

9.10.3.11 void setMaxOutputFileSize (int *size*)

Sets the maximum size of the converted output files.

If the maximum file size is reached, the current trace file is closed and a new one is opened. Passing a negative value to the function will avoid the splitting of the output files.

Siehe auch

loadSettings() (S. 55)

9.10.3.12 int getMaxOutputFileSize ( )

Returns the maximum size of the converted putput files.

For details see **setMaxOutputFileSize()** (S. 56)

9.10.3.13 void setMidnightSplitting (bool *flag*)

Sets the midnight splitting option for file conversion.

If the trace data's time stamps pass midnight, the conversion process by default finishes the current output trace file and starts a new one, so that every output file contains only data of one single day. This behaviour can be disabled passing false to this function.

9.10.3.14 bool isMidnightSplitting ( )

Returns the midnight splitting option of the file conversion.

For details see **setMidnightSplitting()** (S. 56).

9.10.3.15 void useEventTimeInFileNames (bool *flag*)

Sets the rule how to create the output file names.

This function sets the rule how to form the output file name's time span. By default the time span in the file name is adapted to the file's effectively included data. If the EventTimeInFileNames option is active the time span is set to the STARTUP - SHUTDOWN time stamps resp. to the marker time span boundaries selected for data transfer.

9.10.3.16 bool isEventTimeInFileNames ( )

Returns the rule how to create the output file names.

There are two different possibilities how to form the output file name.

Siehe auch

setFileNamesLikeSections()

9.10.3.17 void setChannelToFormat (Channel::BusType *bus,* unsigned int *channelId,* FormatId *fid*)

Sets the channel identified by **Channel::BusType** (S. 50), channel ID to the passed FormatId.

The possible formats are listed in the FormatId header file. Setting a channel to the FormatId NOTRANSFER will deactivate the conversion of this channel's data. This is the default selection for all channels.

Parameter

bus,:	The channel's bus type, see **Channel** (S. 49);
cannelId	channel ID: The index of the bus specific channel starting with 0
fid	possible values: see FormatId

Siehe auch

loadSettings() (S. 55)

Benutzt Channel::toString().

9.10.3.18 FormatId getFormatOfChannel (Channel::BusType *bus,* unsigned int *channelId*)

Returns the format of ChannelTypeId and a channelId.

For more details see **setChannelToFormat()** (S. 57)

9.10.3.19 void setInterruptTracingDuringDownload (bool *flag*)

Sets the option to interrupt data recording during data download.

9.10.3.20 bool isInterruptTracingDuringDownload ( )

Returns wether to interrupt the data recording during data download.

9.10.3.21 void setMarkerTimeSpanForChannelType (Channel::BusType *bus,* int *preTime,* int *postTime*)

Sets the time span around the marker for that the data should be transfered (bus specific). This function is deprecated, use **setMarkerTimeSpanForChannelType(Channel::BusType, LowerBoundType, UpperBoundType, int, int, std::string)** (S. 58) instead.

9.10.3.22 void getMarkerTimeSpanForChannelType (Channel::BusType *bus,* int & *preTime,* int & *postTime*)

Writes the pre and post time for the data of a marker that should be transfered to the passed values (bus specific). This function is deprecated, use **getMarkerTimeSpanForChannelType(Channel::BusType, LowerBoundType&, UpperBoundType&, int&, int&, std::string&)** (S. 59) instead.

9.10.3.23 void setMarkerTimeSpanForChannelType (Channel::BusType *bus,* LowerBoundType *lower,* UpperBoundType *upper,* int *preTime,* int *postTime,* std::string *infoText*)

Sets the time span around the marker for that the data should be transfered (bus specific).

Since BluePiratClient-Library version 3.8.1 there are new possibilities to specifiy the time span around a marker for which the recorded data should be downloaded/converted. Besides the hitherto existing possibility to set a pre time before and a post time after the marker, resp. to select the data time span from the marker's previous startup to the marker's next shutdown or a combination of both, it is now possible to choose another marker as end of the time span. This "end marker" can either be the next marker after the selected one or the next info event with a specified info text. Such info events can be set by the data logger at user defined trigger points when the logger is provided with the "complex trigger" license.

Such marker time spans can be defined for each channel bus type. The avaiable bus types are listed in the documentation of the **Channel** (S. 49) class. Please note that for all MOST channels only one marker time span can be defined. That means, when you call this function twice, one time with bus type MOST_CTRL and once with MOST_SYNC, the latter function call will overwrite the settings of the first one.

Example: The logger triggers a particular bus state and sets the user defined info entry "Bus-State A" to the internal event list. After a while the bus leaves this state and the logger triggers another info event "End bus state A". In between the test driver sets a few marker manually due to any other issues. When reading the data logger, the **EventContainer** (S. 62) contains the following list:

- STARTUP
- MARKER #1
- INFO Bus-State A
- MARKER #2
- MARKER #3
- MARKER #4
- INFO End bus state A

- MARKER #5
- SHUTDOWN
- STARTUP ...

Now you can select the time span between "Bus-State A" and "End bus state A" by setting the tranfer flag of event "Bus-State A" in the **EventContainer** (S. 62) to true and passing the following parameters to the MarkerTimeSpanProperties::setMarkerTimeSpanForChannelType function:

Parameter

bus	The channel bus type for that the marker time span should be set (see **Channel.hh** (S. 134) for all possible bus types), note that for all most channels only one time span can be specified.
lower	The type of the lower boundary (in our example PRETIME, see **TypeDefs.hh** (S. 156) for all possible LowerBoundType)
upper	The type of the upper boundary (in our example NEXT_TEXT_INFO, see **TypeDefs.hh** (S. 156) for all possible UpperBoundType)
preTime	Seconds to transfer before the marker event (in our example 0s)
postTime	Seconds to transfer after the marker event (in our example not relevant)
infoText	The info string of the event that defines the upper boundary (in our exampe "End bus state A")

Benutzt Channel::MOST150_CTRL, Channel::MOST150_ETH, Channel::MOST150_PACKET, Channel::MOST_ASYNC, Channel::MOST_CTRL, Channel::MOST_SYNC, Channel::MOST_SYNC_MAN und Channel::MOST_SYNC_SCAN.

9.10.3.24 void getMarkerTimeSpanForChannelType (Channel::BusType *bus*, LowerBoundType & *lower*, UpperBoundType & *upper*, int & *preTime*, int & *postTime*, std::string & *infoText*)

Writes time span definitions around the marker for that the data should be be transferred to the passed values (bus specific).

For more details see **setMarkerTimeSpanForChannelType(Channel::BusType, LowerBoundType, UpperBoundType, int, int, std::string)** (S. 58)

9.10.3.25 void setAlternativeLoggerName (std::string *name*)

Sets an alternative data logger name.

If this feature is enabled by **useAlternativeLoggerName()** (S. 60), The data logger name set by this function is used instead of the original logger name stored on the data logger to create the output file names.

Siehe auch

loadSettings() (S. 55)

Benutzt BluePiratClientException::INVALID_VALUE.

9.10.3.26 std::string getAlternativeLoggerName ( )

Returns the alternative data logger name.

9.10.3.27 void useAlternativeLoggerName (bool *flag*)

Defines usage of the alternative logger name for the output file names.

The data logger name set by this function is used instead of the original logger name stored on the data logger to create the output file names.

Siehe auch

setAlternativeLoggerName() (S. 59)

9.10.3.28 bool isAlternativeLoggerName ( )

Returns the usage of the alternative logger name for the output file names.

9.10.3.29 void useSubdirectoriesForOutputTraces (bool *flag*)

Defines usage of subdirectories for storage of the converted files.

After conversion the output traces are stored in the target directory set by the **setTargetDirectory()** (S. 55) function. In this directory the trace files are organised in subdirectories that are named by the start time of the included files. This is the default setting. Passing false to the **useSubdirectoriesForOutputTraces()** (S. 60) function will avoid this.

Siehe auch

loadSettings() (S. 55)

9.10.3.30 bool isSubdirectoriesForOutputTraces ( )

Returns the usage of subdirectories for storage of the converted files.

9.11 DataStorageProperties Klassenreferenz

The property class stores settings that consider the storage and protection of recorded trace data.

Öffentliche Methoden

- **DataStorageProperties** ()
  Constructor.
- void **setRingBufferActive** (bool ringBufferActive)
  Enables/disables the ring buffer mode.

- bool **isRingBufferActive** ()
  Returns the ring buffer status.
- void **setProtectMarkerFilesActive** (bool protectMarkerFiles)
  Enables/disables the protection of trace data designated by markers.
- bool **isProtectMarkerFilesActive** ()
  Returns the status of the marker protection.
- void **setRemoveVideoFilesFirstActive** (bool enable)
  Prioritises the deletion of camera data before overwriting old trace data.
- bool **isRemoveVideoFilesFirstActive** ()
  Returns the status of the prioritised camera data deletion.
- virtual bool **setPropertiesToDefaultConfig** ()

9.11.1 Ausführliche Beschreibung

The property class stores settings that consider the storage and protection of recorded trace data.

The **DataStorageProperties** (S. 60) allows enabling and disabling of the ring buffer mode, see **setRingBufferActive()** (S. 61). If the hard drive space runs out and the ring buffer mode is active, the oldest data is overwritten by new data. If it is not active, data logging is just stopped until data is deleted from the data logger. However, data designated by markers can be protected against being overwritten. This feature is enabled via the function **setProtectMarkerFilesActive()** (S. 62). The length of this data block is specified in the property class **MarkerProtectionProperties** (S. 96). For data logger with camera license the deletion of camera data can be prioritised to clear memory space if the hard drive is full.

9.11.2 Beschreibung der Konstruktoren und Destruktoren

9.11.2.1 DataStorageProperties ( )

Constructor.

Constructor that initializes the data storage properties with the default values

- activated ring buffer mode
- marker files protection
- prioritised camera data deletion

Benutzt DataStorageProperties::setPropertiesToDefaultConfig().

9.11.3 Dokumentation der Elementfunktionen

9.11.3.1 void setRingBufferActive (bool *ringBufferActive*)

Enables/disables the ring buffer mode.

9.11.3.2 bool isRingBufferActive ( )

Returns the ring buffer status.

9.11.3.3 void setProtectMarkerFilesActive (bool *protectMarkerFiles*)

Enables/disables the protection of trace data designated by markers.

9.11.3.4 bool isProtectMarkerFilesActive ( )

Returns the status of the marker protection.

9.11.3.5 void setRemoveVideoFilesFirstActive (bool *enable*)

Prioritises the deletion of camera data before overwriting old trace data.

9.11.3.6 bool isRemoveVideoFilesFirstActive ( )

Returns the status of the prioritised camera data deletion.

9.11.3.7 bool setPropertiesToDefaultConfig ( ) [virtual]

Wird benutzt von DataStorageProperties::DataStorageProperties().

9.12 EventContainer Klassenreferenz

Class to store the entries of the event file.

Öffentliche Methoden

- **EventContainer** ()
  *Constructor, creates an empty **EventContainer** (S. 62).*
- ~**EventContainer** ()
  Destructor.
- void **readEventFile** (std::string pathAndFileName)
  *Stores the events from the file in a vector of **EventContainerEntry** (S. 67).*
- void **writeEventFile** (std::string pathAndFileName, bool localTime=false)
  *Writes the current **EventContainer** (S. 62) to a new Event file.*
- void **writeEventOverviewFile** (std::string pathAndFileName)
  Writes all events that are marked for transfer to an overview file.
- unsigned int **getNumEntries** ()
  Returns the current number of container entries.
- void **insertEntry** (**EventContainerEntry** entry, unsigned int index)
  *Inserts a new **EventContainerEntry** (S. 67) at position index.*
- void **appendEntry** (**EventContainerEntry** &entry)
  *appends an **EventContainerEntry** (S. 67) to the end of the container*
- **EventContainerEntry** & **getEntry** (unsigned int index)
  Returns the container entry at position index.

- void **updateEntry** (int index, bool transferFlag)

  *Sets the transferFlag of the **EventContainerEntry** (S. 67) at index to transferFlag.*
- void **deleteEntry** (unsigned int index)

  Deletes the event at index.
- **TimeSpanContainer calculateTimeSpans** (uint64_t preTime, uint64_t postTime, bool lastStartupSelected, bool nextShutdownSelected)

  This function is deprecated. Use the calculateTimeSpans function with Lower- and UpperBoundType as parameters instead.
- **TimeSpanContainer calculateTimeSpans** (uint64_t preTime, uint64_t postTime, **LowerBoundType** lower, **UpperBoundType** upper, std::string endText)

  Calculates the time spans of the marked events using the passed marker boundaries.
- **TimeSpanContainer calculateTimeSpans** (MarkerTimeSpanProperties &properties)

  Calculates the time spans resulting out of the marker time span properties of all channels.
- void **markAllEvents** ()

  Marks all valid events in the event container.
- void **resetTransferFlags** ()

  Resets the transfer flags of the container entries.
- std::string **toString** (int lastEvents=-1)

  Debug function - converts class data to string.
- bool **isSlaveOffsetIncluded** ()

  Returns true if at least one slave offset is included in the event file.
- bool **isSelection** ()

  Returns whether at least one event is selected.
- void **addListener** (**EventContainerListener** *listener)

  Adds a listener to the listener vector.

9.12.1 Ausführliche Beschreibung

Class to store the entries of the event file.

An **EventContainer** (S. 62) stores the entries from the data logger's Events file. It holds the information for which events the data should be transferred. Each event listed in the event file is encapsulated as **EventContainerEntry** (S. 67) and stored in a vector.

Siehe auch

EventContainerEntry (S. 67), **EventType** (S. 19)

9.12.2 Beschreibung der Konstruktoren und Destruktoren

9.12.2.1 EventContainer ( )

Constructor, creates an empty **EventContainer** (S. 62).

9.12.2.2 ~EventContainer ( )

Destructor.

9.12.3 Dokumentation der Elementfunktionen

9.12.3.1 void readEventFile (std::string *pathAndFileName*)

Stores the events from the file in a vector of **EventContainerEntry** (S. 67).

The event file contains startup, shutdown, marker and dataErased events. Each line of the file represents one event with type and time stamp. This function reads the event file line by line and converts each line to an instance of **EventContainerEntry** (S. 67) and stores it in the data member d_containerEntries, a STL vector of **EventContainerEntry** (S. 67). If one line has a corrupt type or time stamp, the function creates an **EventContainerEntry** (S. 67) with Type ERROR_EVENT.

Parameter

pathAndFileName	complete path and file name of the Events file

Benutzt bp::ERROR_EVENT, EventContainerEntry::getEventType(), EventContainerEntry::getTimeStamp(), bp::MARKER, EventContainerEntry::setComment(), EventContainerEntry::setEventType(), EventContainerEntry::setIndex(), EventContainerEntry::setSlaveOffset(), EventContainerEntry::setTimeStamp(), EventContainerEntry::setTransferFlag(), bp::SHUTDOWN und bp::UNKNOWN_EVENT.

9.12.3.2 void writeEventFile (std::string *pathAndFileName,* bool *localTime* = `false`)

Writes the current **EventContainer** (S. 62) to a new Event file.

Method for (over-)writing the event file with the current container entries. This function ignores entries from type ERROR_EVENT. Primarily corrupted lines won't be taken over to the new event file.

Parameter

pathAndFileName	Complete path and file name of the new Events file
localTime	flag whether to write the time stamps in local time

Benutzt bp::DATADOWNLOAD, bp::DATAERASED, bp::INFO, bp::MARKER, bp::NEWTIME, bp::SETTIME, bp::SHUTDOWN, bp::SLAVEOFFSET, bp::SLAVETOMASTER, bp::STARTUP und bp::UNKNOWN_EVENT.

9.12.3.3 void writeEventOverviewFile (std::string *pathAndFileName*)

Writes all events that are marked for transfer to an overview file.

After downloading data the user might need an overview of the sections and marker time stamps he selected to transfer. Therefore the event overview file is written. It contains only the events that were selected by the user in a readable format.

Parameter

pathAndFileName	Complete path and file name if the event overview file

Benutzt bp::DATADOWNLOAD, bp::DATAERASED, bp::INFO, bp::MARKER, bp::SETTIME, bp::SHUTDOWN und bp::STARTUP.

9.12.3.4 unsigned int getNumEntries ( )

Returns the current number of container entries.

This function is deprecated. Use the std::vector::size function instead.

Wird benutzt von EventContainer::toString().

9.12.3.5 void insertEntry (EventContainerEntry *entry,* unsigned int *index*)

Inserts a new **EventContainerEntry** (S. 67) at position *index*.

This function is deprecated. Use the std::vector::insert function instead.

9.12.3.6 void appendEntry (EventContainerEntry & *entry*)

appends an **EventContainerEntry** (S. 67) to the end of the container

This function is deprecated. Use the std::vector::push_back function instead.

9.12.3.7 EventContainerEntry & getEntry (unsigned int *index*)

Returns the container entry at position index.

This function is deprecated. Use the std::vector::operator[] or the std::vector::at() function instead

Wird benutzt von EventContainer::toString().

9.12.3.8 void updateEntry (int *index,* bool *transferFlag*)

Sets the transferFlag of the **EventContainerEntry** (S. 67) at *index* to *transferFlag*.

Selecting a SHUTDOWN event in the container has no effect to a subsequent data transfer. Those events are only used to define the end of a section started by a STARTUP event.

9.12.3.9 void deleteEntry (unsigned int *index*)

Deletes the event at *index*.

This function is deprecated. Use the std::vector::erase function instead.

9.12.3.10 TimeSpanContainer calculateTimeSpans (uint64_t *preTime,* uint64_t *postTime,* bool *lastStartupSelected,* bool *nextShutdownSelected*)

This function is deprecated. Use the calculateTimeSpans function with Lower- and UpperBound-Type as parameters instead.

Calculates the time spans of the marked events using the passed marker boundaries. This function is deprecated. Use the calculateTimeSpans function with Lower- and UpperBoundType as parameters instead

Benutzt bp::LAST_STARTUP, bp::NEXT_SHUTDOWN, bp::POSTTIME und bp::PRETIME.

Wird benutzt von EventContainer::calculateTimeSpans().

9.12.3.11 TimeSpanContainer calculateTimeSpans (uint64_t *preTime,* uint64_t *postTime,* LowerBoundType *lower,* UpperBoundType *upper,* std::string *endText*)

Calculates the time spans of the marked events using the passed marker boundaries.

This function calculates time spans for the events that are selected for transfer. The passed marker boundaries are used for creating a time span for a marker event. The boundaries either result from a fixed time lag around the marker's time stamp or are the last startup / next shutdown before / after the event.

Benutzt TimeSpanContainer::appendTimeSpan(), bp::DATAERASED, bp::INFO, bp::MARKER, bp::NEXT_MARKER_INFO, bp::NEXT_TEXT_INFO, bp::POSTTIME, bp::PRETIME, TimeSpan::setEndTime(), TimeSpan::setStartTime(), bp::SHUTDOWN und bp::STARTUP.

9.12.3.12 TimeSpanContainer calculateTimeSpans (MarkerTimeSpanProperties & *properties*)

Calculates the time spans resulting out of the marker time span properties of all channels.

This function calculates time spans for the events that are selected for transfer. The returned time spans are those resulting from all MarkerTimeSpanProperties of all channels. All overlapping time spans from different channels are merged at the end of this function. The time spans for the video channel are not considered because those over all time spans are only used for conversion.

Siehe auch

calculateTimeSpans(uint64_t preTime, uint64_t postTime, bool lastStartupSelected, bool lastShutdownSelected) (S. 65)

Parameter

properties	Properties where the time span boundaries are definied.

Benutzt TimeSpanContainer::appendTimeSpanContainer(), EventContainer::calculateTimeSpans() und TimeSpanContainer::mergeTimeSpansWithIntersection().

9.12.3.13 void markAllEvents ( )

Marks all valid events in the event container.

Marks all events of type STARTUP, DATAERASED and MARKER. Marking one STARTUP event means marking the entire section between this STARTUP and the next SHUTDOWN

Benutzt bp::DATAERASED, bp::MARKER und bp::STARTUP.

9.12.3.14 void resetTransferFlags ( )

Resets the transfer flags of the container entries.

9.12.3.15 std::string toString (int *lastEvents* = -1)

Debug function - converts class data to string.

Returns the time stamp, event type and the transfer flag of each entry in a formatted string

Benutzt bp::DATADOWNLOAD, bp::DATAERASED, EventContainer::getEntry(), EventContainer::getNumEntries(), bp::INFO, EventContainerEntry::isTransferFlag(), bp::MARKER, bp::NEWTIME, bp::SETTIME, bp::SHUTDOWN, bp::SLAVEOFFSET, bp::SLAVETOMASTER und bp::STARTUP.

9.12.3.16 bool isSlaveOffsetIncluded ( )

Returns true if at least one slave offset is included in the event file.

Benutzt bp::SLAVEOFFSET.

9.12.3.17 bool isSelection ( )

Returns whether at least one event is selected.

9.12.3.18 void addListener (EventContainerListener * *listener*)

Adds a listener to the listener vector.

9.13 EventContainerEntry Klassenreferenz

Class to encapsulate one Events file entry.

Öffentliche Methoden

- **EventContainerEntry** ()
  *Constructor, creates an empty **EventContainerEntry** (S. 67).*
- ~**EventContainerEntry** ()
  Destructor.
- void **setEventType** (**EventType** type)
  Sets the event type.
- const **EventType getEventType** ()
  Returns the event type.
- void **setTimeStamp** (uint64_t timeStamp)
  Sets the time stamp.
- uint64_t **getTimeStamp** ()

Returns the time stamp.

- void **setTransferFlag** (bool flag)

  Sets the transfer flag.
- bool **isTransferFlag** ()

  Returns the transfer flag.
- void **setIndex** (int index)

  Sets the index of the event.
- int **getIndex** ()

  Returns the index of the event.
- void **setSlaveOffset** (int64_t offset)

  Sets the slave offset, only for type SLAVEOFFSET.
- int64_t **getSlaveOffset** ()

  Returns the slave offset, only for type SLAVEOFFSET.
- void **setComment** (std::string comment)

  Sets a comment for entries of type INFO.
- std::string **getComment** ()

  Returns the comment.
- void **setFileSizeOfData** (unsigned size)

  Sets the size of the associated data.
- unsigned **getFileSizeOfData** ()

  Returns the size of the associated data.
- void **setTraceSizeOfData** (unsigned size)

  Sets the size of the uncompressed associated data.
- unsigned **getTraceSizeOfData** ()

  Returns the size of the uncompressed associated data.
- std::string **toString** ()

  Converts the event to a string in Events-File format.
- bool **operator==** (**EventContainerEntry** &entry) const

9.13.1 Ausführliche Beschreibung

Class to encapsulate one Events file entry.

The Event file on the data logger holds the startup, marker, shutdown and dataErased events. Each event has its time stamp. An Instance of the class **EventContainerEntry** (S. 67) stores this time stamp, the event type and the transfer status of the trace data associated with the event. Depending on the data logger's firmware version, the marker events of one data logger are numbered continuously starting at 1. This number is stored as index. All non-marker events are stored with index 0.

Siehe auch

EventContainer (S. 62), **EventType** (S. 19)

9.13.2 Beschreibung der Konstruktoren und Destruktoren

9.13.2.1 EventContainerEntry ( )

Constructor, creates an empty **EventContainerEntry** (S. 67).

9.13.2.2 ~EventContainerEntry ( )

Destructor.

9.13.3 Dokumentation der Elementfunktionen

9.13.3.1 void setEventType (EventType *type*)

Sets the event type.
Wird benutzt von EventContainer::readEventFile().

9.13.3.2 const EventType getEventType ( )

Returns the event type.
Wird benutzt von EventContainer::readEventFile().

9.13.3.3 void setTimeStamp (uint64_t *timeStamp*)

Sets the time stamp.
Wird benutzt von EventContainer::readEventFile().

9.13.3.4 uint64_t getTimeStamp ( )

Returns the time stamp.
Wird benutzt von EventContainer::readEventFile().

9.13.3.5 void setTransferFlag (bool *flag*)

Sets the transfer flag.
Wird benutzt von EventContainer::readEventFile().

9.13.3.6 bool isTransferFlag ( )

Returns the transfer flag.
Wird benutzt von EventContainer::toString().

9.13.3.7 void setIndex (int *index*)

Sets the index of the event.
Wird benutzt von EventContainer::readEventFile().

9.13.3.8 int getIndex ( )

Returns the index of the event.

9.13.3.9 void setSlaveOffset (int64_t *offset*)

Sets the slave offset, only for type SLAVEOFFSET.
Benutzt bp::SLAVEOFFSET.
Wird benutzt von EventContainer::readEventFile().

9.13.3.10 int64_t getSlaveOffset ( )

Returns the slave offset, only for type SLAVEOFFSET.

9.13.3.11 void setComment (std::string *comment*)

Sets a comment for entries of type INFO.
Wird benutzt von EventContainer::readEventFile().

9.13.3.12 std::string getComment ( )

Returns the comment.

9.13.3.13 void setFileSizeOfData (unsigned *size*)

Sets the size of the associated data.

9.13.3.14 unsigned getFileSizeOfData ( )

Returns the size of the associated data.

9.13.3.15 void setTraceSizeOfData (unsigned *size*)

Sets the size of the uncompressed associated data.

9.13.3.16 unsigned getTraceSizeOfData ( )

Returns the size of the uncompressed associated data.

9.13.3.17 std::string toString ( )

Converts the event to a string in Events-File format.

Benutzt bp::DATADOWNLOAD, bp::DATAERASED, bp::INFO, bp::MARKER, bp::NEWTIME, bp::SETTIME, bp::SHUTDOWN, bp::SLAVEOFFSET, bp::SLAVETOMASTER, bp::STARTUP und bp::UNKNOWN_EVENT.

9.13.3.18 bool operator== (EventContainerEntry & *entry*) const

9.14 EventContainerListener Klassenreferenz

Öffentliche Methoden

- virtual void **onStatusReadEventFile** (int percent)

9.14.1 Dokumentation der Elementfunktionen

9.14.1.1 virtual void onStatusReadEventFile (int *percent*) [inline, virtual]

9.15 FlexrayGeneralProperties Klassenreferenz

This class enables to read/write flexray general properries from/to the ConfigConatiner.

Öffentliche Methoden

- **FlexrayGeneralProperties** ()
  Constructor.
- void **setFlexrayConfig** (std::string configFile)
  Called to set flexray configuration.
- std::string **getFlexrayConfig** ()
  Called to get flexray configuain.
- virtual bool **setPropertiesToDefaultConfig** ()

9.15.1 Ausführliche Beschreibung

This class enables to read/write flexray general properries from/to the ConfigConatiner.

Via this class one can set the flexray configuration from a CHI file. Pass the CHI as string to the function **setFlexrayConfig()** (S. 72);

9.15.2 Beschreibung der Konstruktoren und Destruktoren

9.15.2.1 FlexrayGeneralProperties ( )

Constructor.

Constructor that initializes the camera bus property with a default value.

Benutzt setPropertiesToDefaultConfig().

9.15.3 Dokumentation der Elementfunktionen

9.15.3.1 void setFlexrayConfig (std::string *configFile*)

Called to set flexray configuration.

9.15.3.2 std::string getFlexrayConfig ( )

Called to get flexray configuain.

9.15.3.3 bool setPropertiesToDefaultConfig ( ) [virtual]

Wird benutzt von FlexrayGeneralProperties().

9.16 FlexrayProperties Klassenreferenz

This class enables to read/write flexray properries from/to the ConfigConatiner.

Öffentliche Methoden

- **FlexrayProperties** ()
  Constructor,Called implicitly on a definition of the object.
- void **setActiveStatus** (bool activeStatus)
  Called to activate/deactivate the flexray channel.
- bool **isActive** ()
  Called to get flexray active status property.
- void **setName** (std::string name)
  Called to set name property.
- std::string **getName** ()
  Called to get name property.
- void **setChannel** (unsigned int channelId)
  Called to set channel Id.
- unsigned int **getChannel** ()
  Called to get channel Id.
- void **setParameters** (InterfaceParameters param)
  Called to set interface parameters.
- InterfaceParameters **getParameters** ()
  Returns interface parameters.
- virtual bool **setPropertiesToDefaultConfig** ()

9.16.1 Ausführliche Beschreibung

This class enables to read/write flexray properries from/to the ConfigConatiner.

This class is for configuration of the logger's flexray channels. You can set a name, the channel number and activate/deactive the channel. The channel number 0 should be be used for flexray channel A, 1 for channel B.

9.16.2 Beschreibung der Konstruktoren und Destruktoren

9.16.2.1 FlexrayProperties ( )

Constructor,Called implicitly on a definition of the object.
Benutzt setPropertiesToDefaultConfig().

9.16.3 Dokumentation der Elementfunktionen

9.16.3.1 void setActiveStatus (bool *activeStatus*)

Called to activate/deactivate the flexray channel.

9.16.3.2 bool isActive ( )

Called to get flexray active status property.

9.16.3.3 void setName (std::string *name*)

Called to set name property.

9.16.3.4 string getName ( )

Called to get name property.

9.16.3.5 void setChannel (unsigned int *channelId*)

Called to set channel Id.
Wird benutzt von LoggerConfiguration::getFlexrayProperties().

9.16.3.6 unsigned int getChannel ( )

Called to get channel Id.
Wird benutzt von LoggerConfiguration::setFlexrayProperties().

9.16.3.7 void setParameters (InterfaceParameters *param*)

Called to set interface parameters.

9.16.3.8 InterfaceParameters getParameters ( )

Returns interface parameters.

9.16.3.9 bool setPropertiesToDefaultConfig ( ) [virtual]

Wird benutzt von FlexrayProperties().

9.17 GeneralProperties Klassenreferenz

Property class to store general logger settings.

Öffentliche Methoden

- **GeneralProperties** ()
  Constructor.
- void **setLoggerName** (std::string name)
  Sets the data logger name.
- std::string **getLoggerName** ()
  Returns the logger's name.
- void **setTimezoneIndex** (int tzIndex)
  Sets the index of the time zone to apply to the logger.
- int **getTimezoneIndex** ()
  Returns the logger's current time zone index.
- void **setDaylightSavingTimeActive** (bool dst)
  Enables/Disables the automatic daylight saving time for the configured time zone.
- bool **isDaylightSavingTimeActive** ()
  Returns the status of the automatic daylight saving time settimg.
- void **setTraceFileCompressionActive** (bool enable)
  Enables/disables the logger's trace file compression.
- bool **isTraceFileCompressionActive** ()
  Returns trace file compression status.
- virtual bool **setPropertiesToDefaultConfig** ()

9.17.1 Ausführliche Beschreibung

Property class to store general logger settings.

The **GeneralProperties** (S. 74) class stores the name of the data logger. It is not required to set the name of the data logger, however, it is useful to indicate the origin of the trace data this way, since this name is included in the file name of the trace files.

The general properties also include setting the time zone of the data logger and enabling the automatic daylight savings adjustment.

For special licensed logger there is the option to activate/deactivate an trace file compression during recording. This setting is irrelevant for logger without such a license.

9.17.2 Beschreibung der Konstruktoren und Destruktoren

9.17.2.1 GeneralProperties ( )

Constructor.

Constructor that initializes the general properties with the default values

- logger name: 'blue PiraT'
- time zone index: -1,
- automatic summer time conversion: true
- trace file compression: false

Benutzt GeneralProperties::setPropertiesToDefaultConfig().

9.17.3 Dokumentation der Elementfunktionen

9.17.3.1 void setLoggerName (std::string *name*)

Sets the data logger name.

It is not required to set the name of the data logger, however, it is useful to indicate the origin of the trace data this way, since this name is included in the file name of the trace files.

9.17.3.2 std::string getLoggerName ( )

Returns the logger's name.

9.17.3.3 void setTimezoneIndex (int *tzIndex*)

Sets the index of the time zone to apply to the logger.

The passed index must correspond to the index of the TimeZone-Vector returned by the function **BluePiratClient::getTimeZones()** (S. 30).

9.17.3.4 int getTimezoneIndex ( )

Returns the logger's current time zone index.

The returned index corresponds to the TimeZone-Vector returned by the function **BluePiratClient::getTimeZones()** (S. 30).

9.17.3.5 void setDaylightSavingTimeActive (bool *dst*)

Enables/Disables the automatic daylight saving time for the configured time zone.

9.17.3.6 bool isDaylightSavingTimeActive ( )

Returns the status of the automatic daylight saving time settimg.

9.17.3.7 void setTraceFileCompressionActive (bool *enable*)

Enables/disables the logger's trace file compression.

This feature is only available for special licensed logger.

9.17.3.8 bool isTraceFileCompressionActive ( )

Returns trace file compression status.

This feature is only available for special licensed logger.

9.17.3.9 bool setPropertiesToDefaultConfig ( ) [virtual]

Wird benutzt von GeneralProperties::GeneralProperties().

9.18 InterfaceParametersStruct Strukturreferenz

Struct for interface parameters.

Öffentliche Methoden

- **InterfaceParametersStruct** ()

Öffentliche Attribute

- std::string **shortName**
- int **containerId**
- int **version**
- std::string **nameEN**
- std::string **nameDE**
- std::string **type**
- int **rateOfTransfer**
- std::string **comParameter**
- std::string **ipAddress**
- std::string **eculpAddress**
- std::string **netMask**
- int **port**
- std::string **protocol**
- int **debugLevel**

9.18.1 Ausführliche Beschreibung

Struct for interface parameters.

9.18.2 Beschreibung der Konstruktoren und Destruktoren

9.18.2.1 InterfaceParametersStruct ( ) [inline]

Benutzt InterfaceParametersStruct::comParameter, InterfaceParametersStruct::containerId, InterfaceParametersStruct::debugLevel, InterfaceParametersStruct::ecuIpAddress, InterfaceParametersStruct::ipAddress, InterfaceParametersStruct::nameDE, InterfaceParametersStruct::nameEN, InterfaceParametersStruct::netMask, InterfaceParametersStruct::port, InterfaceParametersStruct::protocol, InterfaceParametersStruct::rateOfTransfer, InterfaceParametersStruct::shortName, InterfaceParametersStruct::type und InterfaceParametersStruct::version.

9.18.3 Dokumentation der Datenelemente

9.18.3.1 std::string shortName

Wird benutzt von InterfaceParametersStruct::InterfaceParametersStruct().

9.18.3.2 int containerId

Wird benutzt von InterfaceParametersStruct::InterfaceParametersStruct().

9.18.3.3 int version

Wird benutzt von InterfaceParametersStruct::InterfaceParametersStruct().

9.18.3.4 std::string nameEN

Wird benutzt von InterfaceParametersStruct::InterfaceParametersStruct().

9.18.3.5 std::string nameDE

Wird benutzt von InterfaceParametersStruct::InterfaceParametersStruct().

9.18.3.6 std::string type

Wird benutzt von InterfaceParametersStruct::InterfaceParametersStruct().

9.18.3.7 int rateOfTransfer

Wird benutzt von InterfaceParametersStruct::InterfaceParametersStruct().

9.18.3.8 std::string comParameter

Wird benutzt von InterfaceParametersStruct::InterfaceParametersStruct().

9.18.3.9 std::string ipAddress

Wird benutzt von InterfaceParametersStruct::InterfaceParametersStruct().

9.18.3.10 std::string eculpAddress

Wird benutzt von InterfaceParametersStruct::InterfaceParametersStruct().

9.18.3.11 std::string netMask

Wird benutzt von InterfaceParametersStruct::InterfaceParametersStruct().

9.18.3.12 int port

Wird benutzt von InterfaceParametersStruct::InterfaceParametersStruct().

9.18.3.13 std::string protocol

Wird benutzt von InterfaceParametersStruct::InterfaceParametersStruct().

9.18.3.14 int debugLevel

Wird benutzt von InterfaceParametersStruct::InterfaceParametersStruct().

9.19 LinProperties Klassenreferenz

Property class to store the LIN settings.

Öffentliche Typen

- enum **LinBaudrate** {
  LBAUD_1200, **LBAUD_2400**, **LBAUD_4800**, **LBAUD_9600**,
  LBAUD_19200 }
- enum **LinVersion** { **VERSION_DONT_CARE**, **VERSION_13**, **VERSION_20**, **VERSION_21** }

  List of versions.
- enum **TransceiverType** { **TRANSCEIVER_TYPE_TJA1020** }

  List of transceiver types.

Öffentliche Methoden

- **LinProperties** ()
  Constructor.
- void **setActive** (bool activeStatus)
  Acitvates/deactivates the LIN interface.
- bool **isActive** ()
  Returns the status of the LIN interface.
- void **setName** (std::string name)
  Sets the name of the LIN interface.
- std::string **getName** ()
  Returns the name of the LIN interface.
- void **setWakeupSystemActive** (bool enable)
  Sets the wakeup flag.
- bool **isWakeupSystemActive** ()
  Returns the wakeup flag.
- void **setBaudrate** (**LinBaudrate** baudrate)
  Sets the baud rate of the LIN interface.
- **LinBaudrate getBaudrate** ()
  Returns the baud rate of the LIN interface.
- void **setVersion** (**LinVersion** version)
  Sets the LIN version, currently only VERSION_20 is supported.
- **LinVersion getVersion** ()
  Returns the LIN version.
- void **setTransceiverType** (**TransceiverType** type)
  Sets the transceiver type of the LIN interface, currently only one type supported.
- **TransceiverType getTransceiverType** ()
  Returns the transceiver type of the LIN interface, currently only one type supported.
- void **setChannel** (unsigned int channel)
  Sets the channel of the LIN interface, starting with 0.
- unsigned int **getChannel** ()
  Returns the channel of the LIN interface.
- void **setParameters** (InterfaceParameters param)
  Called to set interface parameters.
- InterfaceParameters **getParameters** ()
  Returns interface parameters.
- virtual bool **setPropertiesToDefaultConfig** ()

9.19.1 Ausführliche Beschreibung

Property class to store the LIN settings.

Each LIN interface can be configured as active or inactive. Only data from active interfaces is recorded. Furthermore, a name can be assigned to each LIN interface, which is later added to the filenames of the trace files for easier identification. The wakeup can be enabled or disabled and the baudrate can be set passing one of the predefined enumeration values, see LinBaudrate. The blue PiraT LIN loggers currently only support the LIN-version 2.0 and the transceiver type "TJA1020". Setting the configuration to another value will have no effect. This functions are for future use.

9.19.2 Dokumentation der Aufzählungstypen

9.19.2.1 enum LinBaudrate

Aufzählungswerte:

LBAUD_1200
LBAUD_2400
LBAUD_4800
LBAUD_9600
LBAUD_19200

9.19.2.2 enum LinVersion

List of versions.

Aufzählungswerte:

VERSION_DONT_CARE
VERSION_13
VERSION_20
VERSION_21

9.19.2.3 enum TransceiverType

List of transceiver types.

Aufzählungswerte:

TRANSCEIVER_TYPE_TJA1020

9.19.3 Beschreibung der Konstruktoren und Destruktoren

9.19.3.1 LinProperties ( )

Constructor.

Constructor that initializes the LIN interface with the default values

- name: "LIN"
- active: true
- baud rate: 9600
- LIN-version: VERSION_DONT_CARE
- transceiver type: TJA1020

Benutzt LinProperties::setPropertiesToDefaultConfig().

9.19.4 Dokumentation der Elementfunktionen

9.19.4.1 void setActive (bool *activeStatus*)

Acitvates/deactivates the LIN interface.

9.19.4.2 bool isActive ( )

Returns the status of the LIN interface.

9.19.4.3 void setName (std::string *name*)

Sets the name of the LIN interface.

The set name is later added to the filenames of the trace files for easier identification

9.19.4.4 std::string getName ( )

Returns the name of the LIN interface.

9.19.4.5 void setWakeupSystemActive (bool *enable*)

Sets the wakeup flag.

This flag defines whether the LIN interface should wake up the logger on bus traffic or not.

9.19.4.6 bool isWakeupSystemActive ( )

Returns the wakeup flag.

Returns whether whether the LIN interface should wake up the logger on bus traffic or not.

9.19.4.7 void setBaudrate (LinBaudrate *baudrate*)

Sets the baud rate of the LIN interface.

Benutzt BluePiratClientException::INVALID_VALUE, LinProperties::LBAUD_1200, LinProperties::LBAUD_19200, LinProperties::LBAUD_2400, LinProperties::LBAUD_4800 und LinProperties::LBAUD_9600.

9.19.4.8 LinProperties::LinBaudrate getBaudrate ( )

Returns the baud rate of the LIN interface.

Benutzt LinProperties::LBAUD_1200, LinProperties::LBAUD_19200, LinProperties::LBAUD_2400, LinProperties::LBAUD_4800 und LinProperties::LBAUD_9600.

9.19.4.9 void setVersion (LinProperties::LinVersion *version*)

Sets the LIN version, currently only VERSION_20 is supported.

9.19.4.10 LinProperties::LinVersion getVersion ( )

Returns the LIN version.

9.19.4.11 void setTransceiverType (LinProperties::TransceiverType *type*)

Sets the transceiver type of the LIN interface, currently only one type supported.

9.19.4.12 LinProperties::TransceiverType getTransceiverType ( )

Returns the transceiver type of the LIN interface, currently only one type supported.

9.19.4.13 void setChannel (unsigned int *channel*)

Sets the channel of the LIN interface, starting with 0.

Wird benutzt von LoggerConfiguration::getLinProperties().

9.19.4.14 unsigned int getChannel ( )

Returns the channel of the LIN interface.

Wird benutzt von LoggerConfiguration::setLinProperties().

9.19.4.15 void setParameters (InterfaceParameters *param*)

Called to set interface parameters.

9.19.4.16 InterfaceParameters getParameters ( )

Returns interface parameters.

9.19.4.17 bool setPropertiesToDefaultConfig ( ) [virtual]

Benutzt LinProperties::TRANSCEIVER_TYPE_TJA1020 und LinProperties::VERSION_DONT_-CARE.

Wird benutzt von LinProperties::LinProperties().

9.20 LoggerConfiguration Klassenreferenz

Class to configure the data logger via client library.

Öffentliche Methoden

- **LoggerConfiguration** ()

  Empty Consturctor.
- ~**LoggerConfiguration** ()
- void **saveToFile** (std::string filePathAndName)

  Saves the current logger configuration to a file.
- void **readFromFile** (std::string filePathAndName)

  Reads the logger configuration from a file.
- void **reset** ()

  Sets all logger properties to its default values.
- void **setSleepProperties** (**SleepProperties** properties)
- **SleepProperties getSleepProperties** ()
- void **setMarkerProperties** (**MarkerProperties** properties)
- **MarkerProperties getMarkerProperties** ()
- void **setGeneralProperties** (**GeneralProperties** properties)
- **GeneralProperties getGeneralProperties** ()
- void **setCascadingProperties** (**CascadingProperties** properties)
- **CascadingProperties getCascadingProperties** ()
- void **setDataStorageProperties** (**DataStorageProperties** properties)
- **DataStorageProperties getDataStorageProperties** ()
- void **setMarkerProtectionProperties** (**MarkerProtectionProperties** properties)
- **MarkerProtectionProperties getMarkerProtectionProperties** ()
- void **setMostProperties** (**MostProperties** properties)
- **MostProperties getMostProperties** ()
- void **setMostFilterProperties** (**MostFilterProperties** properties)
- **MostFilterProperties getMostFilterProperties** ()
- void **setCanProperties** (**CanProperties** properties)
- **CanProperties getCanProperties** (unsigned channel)
- void **setCanFilterProperties** (**CanFilterProperties** properties)
- **CanFilterProperties getCanFilterProperties** (unsigned channel)
- void **setSerialProperties** (**SerialProperties** properties)
- **SerialProperties getSerialProperties** (unsigned port)
- void **setLinProperties** (**LinProperties** properties)
- **LinProperties getLinProperties** (unsigned channel)
- void **setFlexrayProperties** (**FlexrayProperties** properties)
- **FlexrayProperties getFlexrayProperties** (unsigned channel)
- void **setFlexrayGeneralProperties** (**FlexrayGeneralProperties** properties)
- **FlexrayGeneralProperties getFlexrayGeneralProperties** ()
- void **setNetworkConfigProperties** (**NetworkConfigProperties** properties)
- **NetworkConfigProperties getNetworkConfigProperties** ()
- void **setRCVoiceProperties** (**RCVoiceProperties** properties)
- **RCVoiceProperties getRCVoiceProperties** ()

Öffentliche Attribute

- **SleepProperties d_sleepProperties**
- **MarkerProperties d_markerProperties**
- **GeneralProperties d_generalProperties**
- **CascadingProperties d_cascadingProperties**
- **DataStorageProperties d_dataStorageProperties**
- **MarkerProtectionProperties d_markerProtectionProperties**
- **MostProperties d_mostProperties**
- **MostFilterProperties d_mostFilterProperties**
- **FlexrayGeneralProperties d_flexrayGeneralProperties**
- **NetworkConfigProperties d_networkConfigProperties**
- **RCVoiceProperties d_rcVoiceProperties**
- std::vector< **CanProperties** > **d_canPropertiesVector**
- std::vector< **CanFilterProperties** > **d_canFilterPropertiesVector**
- std::vector< **SerialProperties** > **d_serialPropertiesVector**
- std::vector< **LinProperties** > **d_linPropertiesVector**
- std::vector< **FlexrayProperties** > **d_flexrayPropertiesVector**

9.20.1 Ausführliche Beschreibung

Class to configure the data logger via client library.

All settings to configure the data logger via the client library are encapsulated in this class. Use the underlying property classes to modify the logger configuration.

It is strongly recommended to modify the logger's current configuration and set the modified **LoggerConfiguration** (S. 82) back to the **BluePiratClient** (S. 24) instance. The current **LoggerConfiguration** (S. 82) can be received from the **BluePiratClient** (S. 24) instace by calling **BluePiratClient::readConfigurationFromLogger** (S. 30).

br>

Note: All logger channel indices, that can be set and get via the underlying property classes start with zero!

Siehe auch

BluePiratClient (S. 24)

9.20.2 Beschreibung der Konstruktoren und Destruktoren

9.20.2.1 LoggerConfiguration ( )

Empty Consturctor.

9.20.2.2 ~LoggerConfiguration ( )

9.20.3 Dokumentation der Elementfunktionen

9.20.3.1 void saveToFile (std::string *filePathAndName*)

Saves the current logger configuration to a file.

This function doesn't store the CHI file of the flexray configuration. Please store separately.

Benutzt BluePiratClientException::CONFIGURATION_ERROR, LoggerConfiguration::d_canFilterPropertiesVector, LoggerConfiguration::d_canPropertiesVector, LoggerConfiguration::d_cascadingProperties, LoggerConfiguration::d_dataStorageProperties, LoggerConfiguration::d_flexrayPropertiesVector, LoggerConfiguration::d_generalProperties, LoggerConfiguration::d_linPropertiesVector, LoggerConfiguration::d_markerProperties, LoggerConfiguration::d_markerProtectionProperties, LoggerConfiguration::d_mostFilterProperties, LoggerConfiguration::d_mostProperties, LoggerConfiguration::d_networkConfigProperties, LoggerConfiguration::d_rcVoiceProperties, LoggerConfiguration::d_serialPropertiesVector und LoggerConfiguration::d_sleepProperties.

9.20.3.2 void readFromFile (std::string *filePathAndName*)

Reads the logger configuration from a file.

Benutzt BluePiratClientException::CONFIGURATION_ERROR, LoggerConfiguration::d_canFilterPropertiesVector, LoggerConfiguration::d_canPropertiesVector, LoggerConfiguration::d_cascadingProperties, LoggerConfiguration::d_dataStorageProperties, LoggerConfiguration::d_flexrayPropertiesVector, LoggerConfiguration::d_generalProperties, LoggerConfiguration::d_linPropertiesVector, LoggerConfiguration::d_markerProperties, LoggerConfiguration::d_markerProtectionProperties, LoggerConfiguration::d_mostFilterProperties, LoggerConfiguration::d_mostProperties, LoggerConfiguration::d_networkConfigProperties, LoggerConfiguration::d_rcVoiceProperties, LoggerConfiguration::d_serialPropertiesVector und LoggerConfiguration::d_sleepProperties.

9.20.3.3 void reset ( )

Sets all logger properties to its default values.

Benutzt LoggerConfiguration::d_canFilterPropertiesVector, LoggerConfiguration::d_canPropertiesVector, LoggerConfiguration::d_cascadingProperties, LoggerConfiguration::d_dataStorageProperties, LoggerConfiguration::d_flexrayGeneralProperties, LoggerConfiguration::d_flexrayPropertiesVector, LoggerConfiguration::d_generalProperties, LoggerConfiguration::d_linPropertiesVector, LoggerConfiguration::d_markerProperties, LoggerConfiguration::d_markerProtectionProperties, LoggerConfiguration::d_mostFilterProperties, LoggerConfiguration::d_mostProperties, LoggerConfiguration::d_networkConfigProperties, LoggerConfiguration::d_rcVoiceProperties, LoggerConfiguration::d_serialPropertiesVector und LoggerConfiguration::d_sleepProperties.

Wird benutzt von BluePiratClient::readConfigurationFromLogger().

9.20.3.4 void setSleepProperties (SleepProperties *properties*)

Benutzt LoggerConfiguration::d_sleepProperties.

9.20.3.5 SleepProperties getSleepProperties ( )

Benutzt LoggerConfiguration::d_sleepProperties.

9.20.3.6 void setMarkerProperties (MarkerProperties *properties*)

Benutzt LoggerConfiguration::d_markerProperties.

9.20.3.7 MarkerProperties getMarkerProperties ( )

Benutzt LoggerConfiguration::d_markerProperties.

9.20.3.8 void setGeneralProperties (GeneralProperties *properties*)

Benutzt LoggerConfiguration::d_generalProperties.

9.20.3.9 GeneralProperties getGeneralProperties ( )

Benutzt LoggerConfiguration::d_generalProperties.

9.20.3.10 void setCascadingProperties (CascadingProperties *properties*)

Benutzt LoggerConfiguration::d_cascadingProperties.

9.20.3.11 CascadingProperties getCascadingProperties ( )

Benutzt LoggerConfiguration::d_cascadingProperties.

9.20.3.12 void setDataStorageProperties (DataStorageProperties *properties*)

Benutzt LoggerConfiguration::d_dataStorageProperties.

9.20.3.13 DataStorageProperties getDataStorageProperties ( )

Benutzt LoggerConfiguration::d_dataStorageProperties.

9.20.3.14 void setMarkerProtectionProperties (MarkerProtectionProperties *properties*)

Benutzt LoggerConfiguration::d_markerProtectionProperties.

9.20.3.15 MarkerProtectionProperties getMarkerProtectionProperties ( )

Benutzt LoggerConfiguration::d_markerProtectionProperties.

9.20.3.16 void setMostProperties (MostProperties *properties*)

Benutzt LoggerConfiguration::d_mostProperties.

9.20.3.17 MostProperties getMostProperties ( )

Benutzt LoggerConfiguration::d_mostProperties.

9.20.3.18 void setMostFilterProperties (MostFilterProperties *properties*)

Benutzt LoggerConfiguration::d_mostFilterProperties.

9.20.3.19 MostFilterProperties getMostFilterProperties ( )

Benutzt LoggerConfiguration::d_mostFilterProperties.

9.20.3.20 void setCanProperties (CanProperties *properties*)

Benutzt LoggerConfiguration::d_canPropertiesVector und CanProperties::getChannel().

9.20.3.21 CanProperties getCanProperties (unsigned *channel*)

Benutzt LoggerConfiguration::d_canPropertiesVector und CanProperties::setChannel().

9.20.3.22 void setCanFilterProperties (CanFilterProperties *properties*)

Benutzt LoggerConfiguration::d_canFilterPropertiesVector und CanFilterProperties::getChannel().

9.20.3.23 CanFilterProperties getCanFilterProperties (unsigned *channel*)

Benutzt LoggerConfiguration::d_canFilterPropertiesVector und CanFilterProperties::setChannel().

9.20.3.24 void setSerialProperties (SerialProperties *properties*)

Benutzt LoggerConfiguration::d_serialPropertiesVector und SerialProperties::getChannel().

9.20.3.25 SerialProperties getSerialProperties (unsigned *port*)

Benutzt LoggerConfiguration::d_serialPropertiesVector und SerialProperties::setChannel().

9.20.3.26 void setLinProperties (LinProperties *properties*)

Benutzt LoggerConfiguration::d_linPropertiesVector und LinProperties::getChannel().

9.20.3.27 LinProperties getLinProperties (unsigned *channel*)

Benutzt LoggerConfiguration::d_linPropertiesVector und LinProperties::setChannel().

9.20.3.28 void setFlexrayProperties (FlexrayProperties *properties*)

Benutzt LoggerConfiguration::d_flexrayPropertiesVector und FlexrayProperties::getChannel().

9.20.3.29 FlexrayProperties getFlexrayProperties (unsigned *channel*)

Benutzt LoggerConfiguration::d_flexrayPropertiesVector und FlexrayProperties::setChannel().

9.20.3.30 void setFlexrayGeneralProperties (FlexrayGeneralProperties *properties*)

Benutzt LoggerConfiguration::d_flexrayGeneralProperties.

9.20.3.31 FlexrayGeneralProperties getFlexrayGeneralProperties ( )

Benutzt LoggerConfiguration::d_flexrayGeneralProperties.

9.20.3.32 void setNetworkConfigProperties (NetworkConfigProperties *properties*)

Benutzt LoggerConfiguration::d_networkConfigProperties.

9.20.3.33 NetworkConfigProperties getNetworkConfigProperties ( )

Benutzt LoggerConfiguration::d_networkConfigProperties.

9.20.3.34 void setRCVoiceProperties (RCVoiceProperties *properties*)

Benutzt LoggerConfiguration::d_rcVoiceProperties.

9.20.3.35 RCVoiceProperties getRCVoiceProperties ( )

Benutzt LoggerConfiguration::d_rcVoiceProperties.

9.20.4 Dokumentation der Datenelemente

9.20.4.1 SleepProperties d_sleepProperties

Wird benutzt von LoggerConfiguration::getSleepProperties(), LoggerConfiguration::readFromFile(), LoggerConfiguration::reset(), LoggerConfiguration::saveToFile(), LoggerConfiguration::setSleepProperties() und BluePiratClient::writeConfigurationToLogger().

9.20.4.2 MarkerProperties d_markerProperties

Wird benutzt von LoggerConfiguration::getMarkerProperties(), LoggerConfiguration::readFromFile(), LoggerConfiguration::reset(), LoggerConfiguration::saveToFile(), LoggerConfiguration::setMarkerProperties() und BluePiratClient::writeConfigurationToLogger().

9.20.4.3 GeneralProperties d_generalProperties

Wird benutzt von LoggerConfiguration::getGeneralProperties(), LoggerConfiguration::readFromFile(), LoggerConfiguration::reset(), LoggerConfiguration::saveToFile(), LoggerConfiguration::setGeneralProperties() und BluePiratClient::writeConfigurationToLogger().

9.20.4.4 CascadingProperties d_cascadingProperties

Wird benutzt von LoggerConfiguration::getCascadingProperties(), LoggerConfiguration::readFromFile(), LoggerConfiguration::reset(), LoggerConfiguration::saveToFile(), LoggerConfiguration::setCascadingProperties() und BluePiratClient::writeConfigurationToLogger().

9.20.4.5 DataStorageProperties d_dataStorageProperties

Wird benutzt von LoggerConfiguration::getDataStorageProperties(), LoggerConfiguration::readFromFile(), LoggerConfiguration::reset(), LoggerConfiguration::saveToFile(), LoggerConfiguration::setDataStorageProperties() und BluePiratClient::writeConfigurationToLogger().

9.20.4.6 MarkerProtectionProperties d_markerProtectionProperties

Wird benutzt von LoggerConfiguration::getMarkerProtectionProperties(), LoggerConfiguration::readFromFile(), LoggerConfiguration::reset(), LoggerConfiguration::saveToFile(), LoggerConfiguration::setMarkerProtectionProperties() und BluePiratClient::writeConfigurationToLogger().

9.20.4.7 MostProperties d_mostProperties

Wird benutzt von LoggerConfiguration::getMostProperties(), LoggerConfiguration::readFromFile(), LoggerConfiguration::reset(), LoggerConfiguration::saveToFile(), LoggerConfiguration::setMostProperties() und BluePiratClient::writeConfigurationToLogger().

9.20.4.8 MostFilterProperties d_mostFilterProperties

Wird benutzt von LoggerConfiguration::getMostFilterProperties(), LoggerConfiguration::readFromFile(), LoggerConfiguration::reset(), LoggerConfiguration::saveToFile(), LoggerConfiguration::setMostFilterProperties() und BluePiratClient::writeConfigurationToLogger().

9.20.4.9 FlexrayGeneralProperties d_flexrayGeneralProperties

Wird benutzt von LoggerConfiguration::getFlexrayGeneralProperties(), LoggerConfiguration::reset(), LoggerConfiguration::setFlexrayGeneralProperties() und BluePiratClient::writeConfigurationToLogger().

9.20.4.10 NetworkConfigProperties d_networkConfigProperties

Wird benutzt von LoggerConfiguration::getNetworkConfigProperties(), LoggerConfiguration::readFromFile(), LoggerConfiguration::reset(), LoggerConfiguration::saveToFile(), LoggerConfiguration::setNetworkConfigProperties() und BluePiratClient::writeConfigurationToLogger().

9.20.4.11 RCVoiceProperties d_rcVoiceProperties

Wird benutzt von LoggerConfiguration::getRCVoiceProperties(), LoggerConfiguration::readFromFile(), LoggerConfiguration::reset(), LoggerConfiguration::saveToFile(), LoggerConfiguration::setRCVoiceProperties() und BluePiratClient::writeConfigurationToLogger().

9.20.4.12 std::vector<CanProperties> d_canPropertiesVector

Wird benutzt von LoggerConfiguration::getCanProperties(), LoggerConfiguration::readFromFile(), LoggerConfiguration::reset(), LoggerConfiguration::saveToFile(), LoggerConfiguration::setCanProperties() und BluePiratClient::writeConfigurationToLogger().

9.20.4.13 std::vector<CanFilterProperties> d_canFilterPropertiesVector

Wird benutzt von LoggerConfiguration::getCanFilterProperties(), LoggerConfiguration::readFromFile(), LoggerConfiguration::reset(), LoggerConfiguration::saveToFile(), LoggerConfiguration::setCanFilterProperties() und BluePiratClient::writeConfigurationToLogger().

9.20.4.14 std::vector<SerialProperties> d_serialPropertiesVector

Wird benutzt von LoggerConfiguration::getSerialProperties(), LoggerConfiguration::readFromFile(), LoggerConfiguration::reset(), LoggerConfiguration::saveToFile(), LoggerConfiguration::setSerialProperties() und BluePiratClient::writeConfigurationToLogger().

9.20.4.15 std::vector<LinProperties> d_linPropertiesVector

Wird benutzt von LoggerConfiguration::getLinProperties(), LoggerConfiguration::readFromFile(), LoggerConfiguration::reset(), LoggerConfiguration::saveToFile(), LoggerConfiguration::setLinProperties() und BluePiratClient::writeConfigurationToLogger().

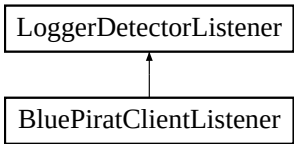
9.20.4.16 std::vector<FlexrayProperties> d_flexrayPropertiesVector

Wird benutzt von LoggerConfiguration::getFlexrayProperties(), LoggerConfiguration::readFromFile(), LoggerConfiguration::reset(), LoggerConfiguration::saveToFile(), LoggerConfiguration::setFlexrayProperties() und BluePiratClient::writeConfigurationToLogger().

9.21 LoggerDetectorListener Klassenreferenz

Base class to listen the LoggerDetector class.

Klassendiagramm für LoggerDetectorListener:



Öffentliche Methoden

- virtual void **onSearchForLoggerStarted** ()
- virtual void **onLoggerDetected** (**bp::LoggerInNetwork** logger)
  Function to notify the listener about logger found in the network.
- virtual bool **onLoggerDisappeared** (**bp::LoggerInNetwork** logger)
  Function to notify the listener about a logger that disappeared from network.
- virtual bool **onSearchForLogger** (int percentCompleted)
  Called to indicate the current progress of getLoggerIpToConnect()
- virtual bool **isLoggerSelected** (std::string &loggerIP)
  Callback to get selected logger.
- virtual void **onSearchForLoggerFinished** ()
  Callback indicate the end of getLoggerIpToConnect()

9.21.1 Ausführliche Beschreibung

Base class to listen the LoggerDetector class.

Since blue PiraT firmware 6.2.1 the data logger can be configured with different network settings (DHCP server, DHCP client, fix IP address). Due to that fact, the blue PiraT Client and library applications can't connect anymore to the standard IP address 192.168.0.231 by default. If there is no logger with the default configuration (DHCP server, standard IP), the client will scan the network for existing loggers. Found loggers are reported via this listener functions. Each detected or disappeared logger will be notified with the appropriate function **onLoggerDetected()** (S. 91) and **onLoggerDisappeared()** (S. 91). The client calls the **isLoggerSelected()** (S. 91) function periodically. If the implementing application wants to connect to a particular logger, it has to implement this function, passing the IP address to the function parameter and returning true.

9.21.2 Dokumentation der Elementfunktionen

9.21.2.1 virtual void onSearchForLoggerStarted ( ) [inline, virtual]

9.21.2.2 virtual void onLoggerDetected (bp::LoggerInNetwork *logger*) [inline, virtual]

Function to notify the listener about logger found in the network.

9.21.2.3 virtual bool onLoggerDisappeared (bp::LoggerInNetwork *logger*) [inline, virtual]

Function to notify the listener about a logger that disappeared from network.

9.21.2.4 virtual bool onSearchForLogger (int *percentCompleted*) [inline, virtual]

Called to indicate the current progress of getLoggerIpToConnect()

9.21.2.5 virtual bool isLoggerSelected (std::string & *loggerIP*) [inline, virtual]

Callback to get selected logger.

9.21.2.6 virtual void onSearchForLoggerFinished ( ) [inline, virtual]

Callback indicate the end of getLoggerIpToConnect()

9.22 LoggerInNetwork Strukturreferenz

Öffentliche Attribute

- std::string **ip**
- std::string **name**
- std::string **mainboard**
- bool **connected**
- std::string **currentUser**

9.22.1 Dokumentation der Datenelemente

9.22.1.1 std::string ip

9.22.1.2 std::string name

9.22.1.3 std::string mainboard

9.22.1.4 bool connected

9.22.1.5 std::string currentUser

9.23 MarkerProperties Klassenreferenz

Property class to store the CAN messages related to the markers.

Öffentliche Methoden

- **MarkerProperties** ()
  Constructor.
- void **setMarkerMessageActive** (bool eventActiveStatus)
  Enables/disables the usage of a CAN message that triggers a marker.
- bool **isMarkerMessageActive** ()
  Returns whether a marker should be triggered by a CAN message.
- void **setMarkerMessageChannel** (int canChannel)
  Sets the CAN channel where to receive the message.
- int **getMarkerMessageChannel** ()
  Returns the CAN channel where to receive the message.
- void **setMarkerMessageCanID** (uint32_t canID)
  Sets the CAN Id of the marker message.
- uint32_t **getMarkerMessageCanID** ()

Returns the CAN Id of the marker message.

- void **setMarkerMessageData** (std::vector< uint8_t > canData)

  Sets the data bytes of the marker message.
- std::vector< uint8_t > **getMarkerMessageData** ()

  Returns the data bytes of the marker message.
- void **setMarkerMessageMask** (std::vector< uint8_t > canMaskData)

  Sets the mask bytes of the marker message.
- std::vector< uint8_t > **getMarkerMessageMask** ()

  Returns the mask bytes of the marker message.
- void **setMarkerMessageDLC** (uint8_t dlc)

  Sets the dlc of the marker message.
- uint8_t **getMarkerMessageDLC** ()

  Returns the dlc of the marker message.
- void **setConfirmationMessageActive** (bool confirmationActiveStatus)

  Enables/disables sending a confirmation CAN message when receiving the marker message.
- bool **isConfirmationMessageActive** ()

  Returns whether a confirmation CAN message should be sent when receiving the marker message.
- void **setConfirmationMessageChannel** (int canChannel)

  Sets the CAN channel where to send the confirmation message.
- int **getConfirmationMessageChannel** ()

  Returns the CAN channel where to send the confirmation message.
- void **setConfirmationMessageCanID** (uint32_t canID)

  Sets the CAN Id of the confirmation message.
- uint32_t **getConfirmationMessageCanID** ()

  Returns the CAN Id of the confirmation message.
- void **setConfirmationMessageData** (std::vector< uint8_t > canData)

  Sets the data bytes of the confirmation message.
- std::vector< uint8_t > **getConfirmationMessageData** ()

  Returns the data bytes of the confirmation message.
- void **setConfirmationMessageDLC** (uint8_t dlc)

  Sets the dlc of the confirmation message.
- uint8_t **getConfirmationMessageDLC** ()

  Returns the dlc of the confirmation message.
- virtual bool **setPropertiesToDefaultConfig** ()

9.23.1 Ausführliche Beschreibung

Property class to store the CAN messages related to the markers.

The blue PiraT data logger offers the possibility to trigger a marker when receiving a certain CAN message, specified by its ID and its data bytes. Additionally it is possible to specify a mask. The trigger condition then is: (Received data byte) BITWISE AND (Mask byte) == (Specified data byte)

If the marker message is received, a CAN message can be sent as the confirmation of setting a marker. The marker message and the confirmation message are specified in this property class.

9.23.2 Beschreibung der Konstruktoren und Destruktoren

9.23.2.1 MarkerProperties ( )

Constructor.
Constructor that initializes the marker properties with the default values

- active marker message: false
- active confirmation message: false

Benutzt MarkerProperties::setPropertiesToDefaultConfig().

9.23.3 Dokumentation der Elementfunktionen

9.23.3.1 void setMarkerMessageActive (bool *eventActiveStatus*)

Enables/disables the usage of a CAN message that triggers a marker.

9.23.3.2 bool isMarkerMessageActive ( )

Returns whether a marker should be triggered by a CAN message.

9.23.3.3 void setMarkerMessageChannel (int *canChannel*)

Sets the CAN channel where to receive the message.
All logger channels start with index 0;.

9.23.3.4 int getMarkerMessageChannel ( )

Returns the CAN channel where to receive the message.

9.23.3.5 void setMarkerMessageCanID (uint32_t *canID*)

Sets the CAN Id of the marker message.

9.23.3.6 uint32_t getMarkerMessageCanID ( )

Returns the CAN Id of the marker message.

9.23.3.7 void setMarkerMessageData (std::vector< uint8_t > *canData*)

Sets the data bytes of the marker message.

9.23.3.8 std::vector< uint8_t > getMarkerMessageData ( )

Returns the data bytes of the marker message.

Wird benutzt von BluePiratClient::writeConfigurationToLogger().

9.23.3.9 void setMarkerMessageMask (std::vector< uint8_t > *canMaskData*)

Sets the mask bytes of the marker message.

9.23.3.10 std::vector< uint8_t > getMarkerMessageMask ( )

Returns the mask bytes of the marker message.

Wird benutzt von BluePiratClient::writeConfigurationToLogger().

9.23.3.11 void setMarkerMessageDLC (uint8_t *dlc*)

Sets the dlc of the marker message.

9.23.3.12 uint8_t getMarkerMessageDLC ( )

Returns the dlc of the marker message.

9.23.3.13 void setConfirmationMessageActive (bool *confirmationActiveStatus*)

Enables/disables sending a confirmation CAN message when receiving the marker message.

9.23.3.14 bool isConfirmationMessageActive ( )

Returns whether a confirmation CAN message should be sent when receiving the marker message.

9.23.3.15 void setConfirmationMessageChannel (int *canChannel*)

Sets the CAN channel where to send the confirmation message.

9.23.3.16 int getConfirmationMessageChannel ( )

Returns the CAN channel where to send the confirmation message.

9.23.3.17 void setConfirmationMessageCanID (uint32_t *canID*)

Sets the CAN Id of the confirmation message.

9.23.3.18 uint32_t getConfirmationMessageCanID ( )

Returns the CAN Id of the confirmation message.

9.23.3.19 void setConfirmationMessageData (std::vector< uint8_t > *canData*)

Sets the data bytes of the confirmation message.

9.23.3.20 std::vector< uint8_t > getConfirmationMessageData ( )

Returns the data bytes of the confirmation message.

9.23.3.21 void setConfirmationMessageDLC (uint8_t *dlc*)

Sets the dlc of the confirmation message.

9.23.3.22 uint8_t getConfirmationMessageDLC ( )

Returns the dlc of the confirmation message.

9.23.3.23 bool setPropertiesToDefaultConfig ( ) [virtual]

Wird benutzt von MarkerProperties::MarkerProperties().

9.24 MarkerProtectionProperties Klassenreferenz

Property class to store the length of the data block to protect around a marker time stamp.

Öffentliche Methoden

- **MarkerProtectionProperties** ()
  Constructor.
- void **setLastStartUp** (bool lastStartUp)
  Call this to let the protected data block begin at the last startup before the marker.
- void **setPreMarkerTime** (int preMarkerTime)
  Call this to let the protected data block begin a certain time before the marker.
- void **setNextShutdown** (bool nextShutdown)
  Call this to let the protected data block end at the next shutdown after the marker.
- void **setPostMarkerTime** (int postMarkerTime)
  Call this to let the protected data block end a certain time after the marker.
- bool **isLastStartUp** ()
  Returns whether the protected data block begins at the last startup before the marker.
- int **getPreMarkerTime** ()

Returns specified pre-marker time .

- bool **isNextShutdown** ()

  Returns whether the protected data block ends at the next shutdown after the marker.

- int **getPostMarkerTime** ()

  Returns specified post-marker time.

- virtual bool **setPropertiesToDefaultConfig** ()

9.24.1 Ausführliche Beschreibung

Property class to store the length of the data block to protect around a marker time stamp.

This settings are relevant if **DataStorageProperties::setProtectMarkerFilesActive** (S. 62) is set to true. The protected data block begins either at a fixed time before the marker time, or at the last startup before the marker. Equivalently, the data block ends a fixed time after the marker time, or at the next shutdown of the data logger.

Note: If a time is given as the data block end, and the data logger shuts down before this time, then the marker data protection ends with the shutdown of the data logger.

9.24.2 Beschreibung der Konstruktoren und Destruktoren

9.24.2.1 MarkerProtectionProperties ( )

Constructor.

Constructor that initializes the properties with the default values

- protect from last startup: false
- protect until next shutdown: false
- time span to protect before the marker: 20s
- time span to protect after the marker: 20s

Benutzt MarkerProtectionProperties::setPropertiesToDefaultConfig().

9.24.3 Dokumentation der Elementfunktionen

9.24.3.1 void setLastStartUp (bool *lastStartUp*)

Call this to let the protected data block begin at the last startup before the marker.

9.24.3.2 void setPreMarkerTime (int *preMarkerTime*)

Call this to let the protected data block begin a certain time before the marker.

The passed passed parameter must be the time in seconds.

9.24.3.3 void setNextShutdown (bool *nextShutdown*)

Call this to let the protected data block end at the next shutdown after the marker.

9.24.3.4 void setPostMarkerTime (int *postMarkerTime*)

Call this to let the protected data block end a certain time after the marker.
The passed passed parameter must be the time in seconds.

9.24.3.5 bool isLastStartUp ( )

Returns whether the protected data block begins at the last startup before the marker.

9.24.3.6 int getPreMarkerTime ( )

Returns specified pre-marker time .

9.24.3.7 bool isNextShutdown ( )

Returns whether the protected data block ends at the next shutdown after the marker.

9.24.3.8 int getPostMarkerTime ( )

Returns specified post-marker time.

9.24.3.9 bool setPropertiesToDefaultConfig ( ) [virtual]

Wird benutzt von MarkerProtectionProperties::MarkerProtectionProperties().

9.25 MostFilterProperties::MostFilter Strukturreferenz

Strcut that stores a set of MOST parameters.

Öffentliche Attribute

- int **senderID**
- int **receiverID**
- int **functionBlockID**
- int **instanceID**
- int **functionID**
- int **opType**

9.25.1 Ausführliche Beschreibung

Strcut that stores a set of MOST parameters.

Parameters that shouldn't care must be set to -1

9.25.2 Dokumentation der Datenelemente

9.25.2.1 int senderID

9.25.2.2 int receiverID

9.25.2.3 int functionBlockID

9.25.2.4 int instanceID

9.25.2.5 int functionID

9.25.2.6 int opType

9.26 MostFilterProperties Klassenreferenz

Property class to store most filter.

Klassen

- struct **MostFilter**
  Strcut that stores a set of MOST parameters.

Öffentliche Methoden

- **MostFilterProperties** ()
  Constructor.
- void **setMostFilters** (std::vector< **MostFilter** > filterElements)
  Sets the MOST filters.
- std::vector< **MostFilter** > **getMostFilters** ()
  Returns the MOST filters.
- void **setActive** (bool status)
  Activates/deactivates the MOST filters.
- bool **isActive** ()
  Returns the active status.
- virtual bool **setPropertiesToDefaultConfig** ()

9.26.1 Ausführliche Beschreibung

Property class to store most filter.

This property class stores a list of filter entries, where each entry specifies a set of message parameters. A MOST message should pass if all parameters of one entry match. Parameters that don't care must be set to -1. The set of parameters consists of:

- sender ID
- receiver ID
- function block ID
- instance ID
- function ID
- and opType

9.26.2 Beschreibung der Konstruktoren und Destruktoren

9.26.2.1 MostFilterProperties ( )

Constructor.

Constructor that initializes the MOST filter properties with the default values.

Benutzt MostFilterProperties::setPropertiesToDefaultConfig().

9.26.3 Dokumentation der Elementfunktionen

9.26.3.1 void setMostFilters (std::vector< MostFilter > *filterElements*)

Sets the MOST filters.

The MOST filters are a set of **MostFilter** (S. 98) objects stored as vector.

9.26.3.2 std::vector< MostFilterProperties::MostFilter > getMostFilters ( )

Returns the MOST filters.

The MOST filters are a returned as vector of **MostFilter** (S. 98) objects.

9.26.3.3 void setActive (bool *status*)

Activates/deactivates the MOST filters.

9.26.3.4 bool isActive ( )

Returns the active status.

9.26.3.5 bool setPropertiesToDefaultConfig ( ) [virtual]

Wird benutzt von MostFilterProperties::MostFilterProperties().

9.27 MostProperties Klassenreferenz

Property class to store the MOST configuratio settings.

Öffentliche Methoden

- **MostProperties** ()
  Constructor.
- void **setAsyncChannelActive** (bool asyncChannelTraceActive)
  Enables/disables the recording of the MOST asynchronous channel.
- bool **isAsyncChannelActive** ()
  Returns the status of the MOST asynchronous channel.
- void **setAsyncChannelArbitrationValue** (int arbitrationValue)
  Sets the highest accepted arbitration value for asynchronous messages.
- int **getAsyncChannelArbitrationValue** ()
  Returns the highest accepted arbitration value for asynchronous messages.
- void **setDataRegenerationActive** (bool regenerationMode)
  Activates/deactivates the MOST signal regeneration mode.
- bool **isDataRegenerationActive** ()
  Returns the status of the MOST signal regeneration.
- bool **isRecordParityErrorMessagesActive** ()
  Returns record parity error messages status.
- void **setRecordParityErrorMessages** (bool record)
  Called to set record parity error messages status.
- bool **isRecordInvalidArbitrationMessagesActive** ()
  Returns record invalid arbitration messages status.
- void **setRecordInvalidArbitrationMessages** (bool record)
  Called to set record invalid arbitration messages.
- virtual bool **setPropertiesToDefaultConfig** ()
  Returns status of training feature.
- void **setName** (std::string name)
  Called to set name property.
- std::string **getName** ()
  Called to get name property.
- void **setParameters** (InterfaceParameters param)
  Called to set interface parameters.
- InterfaceParameters **getParameters** ()
  Returns interface parameters.

9.27.1 Ausführliche Beschreibung

Property class to store the MOST configuratio settings.

With this properties the MOST regeneration mode can be enabled or disabled. It is also possible to enable or disable the logging of the MOST asynchronous channel and to specify the highest accepted arbitration value for asynchronous messages, which is set to 0x1F by default. All messages with a higher value are not recorded. For more information see the user's manual of the blue PiraT data logger.

9.27.2 Beschreibung der Konstruktoren und Destruktoren

9.27.2.1 MostProperties ( )

Constructor.

Constructor that initializes the MOST properties with the default values

- async channel active: true
- highest arbitration value: 0x1F
- MOST regeneration active: true

Benutzt MostProperties::setPropertiesToDefaultConfig().

9.27.3 Dokumentation der Elementfunktionen

9.27.3.1 void setAsyncChannelActive (bool *asyncChannelTraceActive*)

Enables/disables the recording of the MOST asynchronous channel.

9.27.3.2 bool isAsyncChannelActive ( )

Returns the status of the MOST asynchronous channel.

9.27.3.3 void setAsyncChannelArbitrationValue (int *arbitrationValue*)

Sets the highest accepted arbitration value for asynchronous messages.

The highest accepted arbitration value is set to 0x1F by default. All messages with a higher value are not recorded. For more information see the user's manual of the blue PiraT data logger.

9.27.3.4 int getAsyncChannelArbitrationValue ( )

Returns the highest accepted arbitration value for asynchronous messages.

9.27.3.5 void setDataRegenerationActive (bool *regenerationMode*)

Activates/deactivates the MOST signal regeneration mode.

9.27.3.6 bool isDataRegenerationActive ( )

Returns the status of the MOST signal regeneration.

9.27.3.7 bool isRecordParityErrorMessagesActive ( )

Returns record parity error messages status.

9.27.3.8 void setRecordParityErrorMessages (bool *record*)

Called to set record parity error messages status.

9.27.3.9 bool isRecordInvalidArbitrationMessagesActive ( )

Returns record invalid arbitration messages status.

9.27.3.10 void setRecordInvalidArbitrationMessages (bool *record*)

Called to set record invalid arbitration messages.

9.27.3.11 bool setPropertiesToDefaultConfig ( ) [virtual]

Returns status of training feature.
Called to set the status of training feature
Wird benutzt von MostProperties::MostProperties().

9.27.3.12 void setName (std::string *name*)

Called to set name property.

9.27.3.13 std::string getName ( )

Called to get name property.

9.27.3.14 void setParameters (InterfaceParameters *param*)

Called to set interface parameters.

9.27.3.15 InterfaceParameters getParameters ( )

Returns interface parameters.

9.28 NetworkConfigProperties Klassenreferenz

Property class to store the dhcp settings.

Öffentliche Typen

- enum **DHCPMode** { **DHCP_OFF**, **DHCP_CLIENT**, **DHCP_SERVER** }
  List of DHCP mode.

Öffentliche Methoden

- **NetworkConfigProperties** ()
  Constructor.
- void **setIP** (std::string ip)
  Sets the network ip.
- void **setNetmask** (std::string netMask)
  Sets the network mask.
- void **setDHCPMode** (**NetworkConfigProperties::DHCPMode** dhcpMode)
  Sets the DHCP mode.
- std::string **getIP** ()
  Gets the network ip.
- std::string **getNetmask** ()
  Gets the network mask.
- **NetworkConfigProperties::DHCPMode getDHCPMode** ()
  Gets the DHCP mode.
- virtual bool **setPropertiesToDefaultConfig** ()

Statische öffentliche Attribute

- static **DHCPMode d_dhcpModeOnConfigReading**

9.28.1 Ausführliche Beschreibung

Property class to store the dhcp settings.

9.28.2 Dokumentation der Aufzählungstypen

9.28.2.1 enum DHCPMode

List of DHCP mode.

Pass one of these to the setDHCPMode function.

Aufzählungswerte:

DHCP_OFF
DHCP_CLIENT
DHCP_SERVER

9.28.3 Beschreibung der Konstruktoren und Destruktoren

9.28.3.1 NetworkConfigProperties ( )

Constructor.
It is used to show warning message on writing the config to logger.
Benutzt NetworkConfigProperties::setPropertiesToDefaultConfig().

9.28.4 Dokumentation der Elementfunktionen

9.28.4.1 void setIP (std::string *ip*)

Sets the network ip.

9.28.4.2 void setNetmask (std::string *netMask*)

Sets the network mask.

9.28.4.3 void setDHCPMode (NetworkConfigProperties::DHCPMode *dhcpMode*)

Sets the DHCP mode.

9.28.4.4 std::string getIP ( )

Gets the network ip.
Wird benutzt von BluePiratClient::writeConfigurationToLogger().

9.28.4.5 std::string getNetmask ( )

Gets the network mask.
Wird benutzt von BluePiratClient::writeConfigurationToLogger().

9.28.4.6 NetworkConfigProperties::DHCPMode getDHCPMode ( )

Gets the DHCP mode.
Wird benutzt von BluePiratClient::writeConfigurationToLogger().

9.28.4.7 bool setPropertiesToDefaultConfig ( ) [virtual]

Benutzt NetworkConfigProperties::DHCP_SERVER.
Wird benutzt von NetworkConfigProperties::NetworkConfigProperties().

9.28.5 Dokumentation der Datenelemente

9.28.5.1 NetworkConfigProperties::DHCPMode d_dhcpModeOnConfigReading [static]

Constructor that initializes the dhcp properties with the default values

- ip : 192.168.0.231
- netmask: 255.255.255.0
- dhcp: off

9.29 RCVoiceProperties Klassenreferenz

Property class to store the RCVoice configuration settings.

Öffentliche Methoden

- **RCVoiceProperties** ()
  Constructor.
- ~**RCVoiceProperties** ()
  Destructor.
- void **setRecordingLength** (uint16_t recLength)
  Sets the remote control's voice note recording length.
- uint16_t **getRecordingLength** ()
  Returns the recording length of the voice notes.
- virtual bool **setPropertiesToDefaultConfig** ()
  Called to set Properties to default settings.

9.29.1 Ausführliche Beschreibung

Property class to store the RCVoice configuration settings.

With this properties the RCVoice can be enabled or disabled. It is also possible to configure the recording length of a RC voice note.

9.29.2 Beschreibung der Konstruktoren und Destruktoren

9.29.2.1 RCVoiceProperties ( )

Constructor.

Constructor that initializes the MOST properties with the default values

- async channel active: true
- highest arbitration value: 0x1F
- MOST regeneration active: true

Benutzt RCVoiceProperties::setPropertiesToDefaultConfig().

9.29.2.2 ~RCVoiceProperties ( )

Destructor.

9.29.3 Dokumentation der Elementfunktionen

9.29.3.1 void setRecordingLength (uint16_t *recLength*)

Sets the remote control's voice note recording length.

9.29.3.2 uint16_t getRecordingLength ( )

Returns the recording length of the voice notes.

9.29.3.3 bool setPropertiesToDefaultConfig ( ) [virtual]

Called to set Properties to default settings.

Wird benutzt von RCVoiceProperties::RCVoiceProperties().

9.30 TimeZone::RegTimezoneInformation Strukturreferenz

Öffentliche Attribute

- long **Bias**
- long **StandardBiasAs**
- long **DaylightBias**
- SYSTEMTIME **StandardDate**
- SYSTEMTIME **DaylightDate**

9.30.1 Dokumentation der Datenelemente

9.30.1.1 long Bias

9.30.1.2 long StandardBiasAs

9.30.1.3 long DaylightBias

9.30.1.4 SYSTEMTIME StandardDate

9.30.1.5 SYSTEMTIME DaylightDate

9.31 SerialProperties Klassenreferenz

Property class to store the serial interface settings.

Öffentliche Typen

- enum **SerialProtocol** { **PROTOCOL_NONE**, **PROTOCOL_TRACECLIENT**, **PROTOCOL_GNLOGGER**, **PROTOCOL_DLTBMW** }

  List of serial protocols.
- enum **TransceiverType** { **TRANSCEIVER_TYPE_RS232**, **TRANSCEIVER_TYPE_RS422** }

  List of transceiver types.
- enum **SerialBaudrate** {
  SBAUD_110, **SBAUD_300**, **SBAUD_1200**, **SBAUD_2400**,
  SBAUD_4800, **SBAUD_9600**, **SBAUD_19200**, **SBAUD_38400**,
  SBAUD_57600, **SBAUD_115200** }

  List of serial baud rates.
- enum **Parity** { **NONE**, **EVEN**, **ODD** }

  List of parity choices.

Öffentliche Methoden

- **SerialProperties** ()

  Constructor.
- void **setName** (std::string name)

  Sets the name of the serial interface.
- void **setTransceiverType** (**TransceiverType** transceiverType)

  Sets the transceiver type.
- void **setBaudRate** (**SerialBaudrate** baudrate)

  Sets the baud rate.
- void **setNumDataBits** (int numDatabits)

  Sets the number of data bits.
- void **setNumStopBits** (int numStopbits)

  Sets the number of stop bits.
- void **setParity** (**Parity** parity)

  Sets the parity.
- void **setProtocol** (**SerialProtocol** protocol)

  Sets the protocol.
- void **setChannel** (int channel)

  Sets the channel of the serial interface.
- void **setEcuId** (std::string ecuId)

  Sets the ecuId , It is only used if (dltBMwLogLicense && featureDLTBMWLogging = true)
- std::string **getName** ()

  Returns the name of the serial interface.
- **TransceiverType getTransceiverType** ()

  Returns the transceiver type.
- **SerialBaudrate getBaudRate** ()

  Returns the baud rate.
- int **getNumDataBits** ()

  Returns the number of data bits.
- int **getNumStopBits** ()

Returns the number of stop bits.

- **Parity getParity** ()

  Returns the parity.
- **SerialProtocol getProtocol** ()

  Returns the protocol.
- int **getChannel** ()

  Returns the channel.
- std::string **getEcuId** ()

  Returns the ecuId , It is only used if (dltBMwLogLicense && featureDLTBMWLogging = true)
- void **setParameters** (InterfaceParameters param)

  Called to set interface parameters.
- InterfaceParameters **getParameters** ()

  Returns interface parameters.
- virtual bool **setPropertiesToDefaultConfig** ()

9.31.1 Ausführliche Beschreibung

Property class to store the serial interface settings.

The usual properties of the serial port are configurable: Baudrate, number of data bits, number of stop bits and the parity setting. It is possible to assign a name to the serial ports as well. For data loggers with the option "RS422", the transceiver type of the serial port can be switched between RS232 and RS422.

9.31.2 Dokumentation der Aufzählungstypen

9.31.2.1 enum SerialProtocol

List of serial protocols.

Pass one of these to the setProtocol function.

Aufzählungswerte:

PROTOCOL_NONE

PROTOCOL_TRACECLIENT

PROTOCOL_GNLOGGER

PROTOCOL_DLTBMW

9.31.2.2 enum TransceiverType

List of transceiver types.

Aufzählungswerte:

TRANSCEIVER_TYPE_RS232

TRANSCEIVER_TYPE_RS422

9.31.2.3 enum SerialBaudrate

List of serial baud rates.

Pass one of these to the setBaudRate function.

Aufzählungswerte:

SBAUD_110
SBAUD_300
SBAUD_1200
SBAUD_2400
SBAUD_4800
SBAUD_9600
SBAUD_19200
SBAUD_38400
SBAUD_57600
SBAUD_115200

9.31.2.4 enum Parity

List of parity choices.

Pass one of these to the setParity function.

Aufzählungswerte:

NONE
EVEN
ODD

9.31.3 Beschreibung der Konstruktoren und Destruktoren

9.31.3.1 SerialProperties ( )

Constructor.

Constructor that initializes the serial properties with the default values

- transceiver type: RS232
- Baud rate: 115200
- number of data bits: 8
- number of stop bits: 1
- parity: event
- protocol: none
- ecuId: ECU<channel number>

Benutzt SerialProperties::setPropertiesToDefaultConfig().

9.31.4 Dokumentation der Elementfunktionen

9.31.4.1 void setName (std::string *name*)

Sets the name of the serial interface.

9.31.4.2 void setTransceiverType (TransceiverType *transceiverType*)

Sets the transceiver type.

9.31.4.3 void setBaudRate (SerialBaudrate *baudrate*)

Sets the baud rate.

Benutzt BluePiratClientException::INVALID_VALUE, SerialProperties::SBAUD_110, SerialProperties::SBAUD_115200, SerialProperties::SBAUD_1200, SerialProperties::SBAUD_19200, SerialProperties::SBAUD_2400, SerialProperties::SBAUD_300, SerialProperties::SBAUD_38400, SerialProperties::SBAUD_4800, SerialProperties::SBAUD_57600 und SerialProperties::SBAUD_9600.

9.31.4.4 void setNumDataBits (int *numDatabits*)

Sets the number of data bits.

9.31.4.5 void setNumStopBits (int *numStopbits*)

Sets the number of stop bits.

9.31.4.6 void setParity (SerialProperties::Parity *parity*)

Sets the parity.

Benutzt SerialProperties::EVEN, BluePiratClientException::INVALID_VALUE, SerialProperties::NONE und SerialProperties::ODD.

9.31.4.7 void setProtocol (SerialProtocol *protocol*)

Sets the protocol.

9.31.4.8 void setChannel (int *channel*)

Sets the channel of the serial interface.

Wird benutzt von LoggerConfiguration::getSerialProperties().

9.31.4.9 void setEcuId (std::string *ecuId*)

Sets the ecuId , It is only used if (dltBMwLogLicense && featureDLTBMWLogging = true)

9.31.4.10 std::string getName ( )

Returns the name of the serial interface.

9.31.4.11 SerialProperties::TransceiverType getTransceiverType ( )

Returns the transceiver type.

9.31.4.12 SerialProperties::SerialBaudrate getBaudRate ( )

Returns the baud rate.

Benutzt SerialProperties::SBAUD_110, SerialProperties::SBAUD_115200, SerialProperties::SBAUD_1200, SerialProperties::SBAUD_19200, SerialProperties::SBAUD_2400, SerialProperties::SBAUD_300, SerialProperties::SBAUD_38400, SerialProperties::SBAUD_4800, SerialProperties::SBAUD_57600 und SerialProperties::SBAUD_9600.

9.31.4.13 int getNumDataBits ( )

Returns the number of data bits.

9.31.4.14 int getNumStopBits ( )

Returns the number of stop bits.

9.31.4.15 SerialProperties::Parity getParity ( )

Returns the parity.

Benutzt SerialProperties::EVEN, SerialProperties::NONE und SerialProperties::ODD.

9.31.4.16 SerialProperties::SerialProtocol getProtocol ( )

Returns the protocol.

9.31.4.17 int getChannel ( )

Returns the channel.

Wird benutzt von LoggerConfiguration::setSerialProperties().

9.31.4.18 std::string getEcuId ( )

Returns the ecuId , It is only used if (dltBMwLogLicense && featureDLTBMWLogging = true)

Returns ECU id property.

9.31.4.19 void setParameters (InterfaceParameters *param*)

Called to set interface parameters.

9.31.4.20 InterfaceParameters getParameters ( )

Returns interface parameters.

9.31.4.21 bool setPropertiesToDefaultConfig ( ) [virtual]

Benutzt SerialProperties::PROTOCOL_NONE und SerialProperties::TRANSCEIVER_TYPE_RS232.

Wird benutzt von SerialProperties::SerialProperties().

9.32 SleepProperties Klassenreferenz

Property class to store the power management settings.

Öffentliche Methoden

- **SleepProperties** ()
  Constructor.
- void **setSleepMessageTimeout** (uint16_t sleepmessagetimeout)
  Sets the data time out in seconds.
- uint16_t **getSleepMessageTimeout** ()
  Returns the data time out in seconds.
- void **setSleepNetworkTimeout** (uint16_t sleepNetworkTimeout)
  Sets the network time out in seconds.
- uint16_t **getSleepNetworkTimeout** ()
  Returns the network time out in seconds.
- void **setDeactivateAutomaticShutdown** (bool deactivateAutomaticShutdown)
  Deactivates/Activates the automatic shoutdown of the data logger.
- bool **isDeactivateAutomaticShutdown** ()
  Returns status of the logger's automatic shutdown.
- virtual bool **setPropertiesToDefaultConfig** ()

9.32.1 Ausführliche Beschreibung

Property class to store the power management settings.

The power **SleepProperties** (S. 113) provide the setup of the shutdown condition of the data logger. Currently, the only condition is the data timeout. If the data logger does not receive any data (i.e., no serial data, no CAN message, and MOST light is off) during this timeout, it shuts down and enters standby mode. For the case of a connected network cable with active link, a

separate value for the timeout can be specified. It is also possible to deactivate the automatic standby. For more information about the power management, see the blue PiraT User's Manual.

Warning: Automatic standby should only be deactivated if the data logger is connected to a sufficient power supply. If the data logger is supplied by the car battery, the battery might be discharged quickly, resulting in insufficient charge for crank-up of the engine.

9.32.2 Beschreibung der Konstruktoren und Destruktoren

9.32.2.1 SleepProperties ( )

Constructor.

Constructor that initializes the power management settings with the default values

- message time out: 180s
- network time out: 3600s
- inactive automatic shutdown: false

Benutzt SleepProperties::setPropertiesToDefaultConfig().

9.32.3 Dokumentation der Elementfunktionen

9.32.3.1 void setSleepMessageTimeout (uint16_t *sleepmessagetimeout*)

Sets the data time out in seconds.

If the data logger does not receive any message (i.e., no serial data, no CAN message, and MOST light is off) during this timeout, it shuts down and enters standby mode.

9.32.3.2 uint16_t getSleepMessageTimeout ( )

Returns the data time out in seconds.

9.32.3.3 void setSleepNetworkTimeout (uint16_t *sleepNetworkTimeout*)

Sets the network time out in seconds.

For the case of a connected network cable with active link, a separate value for the timeout can be specified. For example, if a PC/Laptop is connected, the value could be increased to ensure that the data logger stays active a longer time.

9.32.3.4 uint16_t getSleepNetworkTimeout ( )

Returns the network time out in seconds.

9.32.3.5 void setDeactivateAutomaticShutdown (bool *deactivateAutomaticShutdown*)

Deactivates/Activates the automatic shoutdown of the data logger.

Warning: Automatic standby should only be deactivated if the data logger is connected to a sufficient power supply. If the data logger is supplied by the car battery, the battery might be discharged quickly, resulting in insufficient charge for crank-up of the engine.

9.32.3.6 bool isDeactivateAutomaticShutdown ( )

Returns status of the logger's automatic shutdown.

9.32.3.7 bool setPropertiesToDefaultConfig ( ) [virtual]

Wird benutzt von SleepProperties::SleepProperties().

9.33 TimeSpan Klassenreferenz

Class to store a time span.

Öffentliche Methoden

- **TimeSpan** ()
  *Constructor, creates an empty **TimeSpan** (S. 115).*
- **TimeSpan** (uint64_t start, uint64_t end)
  *Constructor, creates a **TimeSpan** (S. 115) and sets the start and end time.*
- ~**TimeSpan** ()
  Destructor.
- void **setStartTime** (uint64_t time)
  Sets the start time of the time span to the passed usec value.
- uint64_t **getStartTime** ()
  Returns the time span's start time.
- void **setEndTime** (uint64_t time)
  Sets the end time of the time span to the passed usec value.
- uint64_t **getEndTime** ()
  Returns the time span's end time.
- bool **doesIntersect** (**TimeSpan** timeSpan, bool includeBorder=false)
  Returns true if the time span overlaps with the passed timeSpan.
- bool **includes** (uint64_t timeStamp)
  Returns true if the time span includes the passed timeStamp.
- void **convertToLoggerTimeZone** ()
  Converts the time span to the logger's time zone.

9.33.1 Ausführliche Beschreibung

Class to store a time span.

Stores a time span with given start time and end time. Both values are definied as usec since midnight 01.01.1970 (UTC)

Siehe auch

TimeSpanContainer (S. 117)

9.33.2 Beschreibung der Konstruktoren und Destruktoren

9.33.2.1 TimeSpan ( )

Constructor, creates an empty **TimeSpan** (S. 115).

9.33.2.2 TimeSpan (uint64_t *start*, uint64_t *end*)

Constructor, creates a **TimeSpan** (S. 115) and sets the start and end time.

9.33.2.3 ~TimeSpan ( )

Destructor.

9.33.3 Dokumentation der Elementfunktionen

9.33.3.1 void setStartTime (uint64_t *time*)

Sets the start time of the time span to the passed usec value.

Wird benutzt von EventContainer::calculateTimeSpans().

9.33.3.2 uint64_t getStartTime ( )

Returns the time span's start time.

Wird benutzt von TimeSpanContainer::appendTimeSpan(), TimeSpan::doesIntersect() und TimeSpanContainer::getIncludedTimeSpan().

9.33.3.3 void setEndTime (uint64_t *time*)

Sets the end time of the time span to the passed usec value.

Wird benutzt von EventContainer::calculateTimeSpans().

9.33.3.4 uint64_t getEndTime ( )

Returns the time span's end time.

Wird benutzt von TimeSpanContainer::appendTimeSpan() und TimeSpan::doesIntersect().

9.33.3.5 bool doesIntersect (TimeSpan *timeSpan*, bool *includeBorder* = `false`)

Returns true if the time span overlaps with the passed *timeSpan*.

Benutzt TimeSpan::getEndTime() und TimeSpan::getStartTime().

9.33.3.6 bool includes (uint64_t *timeStamp*)

Returns true if the time span includes the passed *timeStamp*.

9.33.3.7 void convertToLoggerTimeZone ( )

Converts the time span to the logger's time zone.

This functions converts the time spans to the according logger time spans using the logger's time zone. For example: The current time span was created from the local system time at 15:00:00. The PC calculates the UTC seconds since 1970 considering the systems time zone. If you need the UTC seconds of todays 15:00:00 o'clock, assuming that the time zone is that of the logger, use this function to convert.

9.34 TimeSpanContainer Klassenreferenz

Class to store a number of TimeSpans.

Öffentliche Methoden

- **TimeSpanContainer** ()
  Constructor, creates an empty ***TimeSpanContainer*** *(S. 117).*
- ~**TimeSpanContainer** ()
  Destructor.
- unsigned int **getNumEntries** ()
  Returns the size of the intern ***TimeSpan*** *(S. 115) vector.*
- **TimeSpan getEntry** (unsigned int index)
  Returns the timeSpan at index.
- void **appendTimeSpan** (**TimeSpan** timeSpan)
  Appends a time span to the container.
- void **appendTimeSpanContainer** (**TimeSpanContainer** container)
  Appends a time span container.
- void **mergeTimeSpansWithIntersection** (bool logMerging=true)
  Merges time spans that overlap each other.
- **TimeSpan getIncludedTimeSpan** (**TimeSpan** timespan)

Returns the time span that is included in the passed one.

- void **deleteAllEntries** ()

  Deletes all entries.

- std::string **toString** ()

  Debug function - converts class data to string.

9.34.1 Ausführliche Beschreibung

Class to store a number of TimeSpans.

The class **TimeSpanContainer** (S. 117) stores a number of time spans from type **TimeSpan** (S. 115). A **TimeSpanContainer** (S. 117) provides access to a vector of TimeSpans plus some additional processing functions.

Siehe auch

TimeSpan (S. 115)

9.34.2 Beschreibung der Konstruktoren und Destruktoren

9.34.2.1 TimeSpanContainer ( )

Constructor, creates an empty **TimeSpanContainer** (S. 117).

9.34.2.2 ~TimeSpanContainer ( )

Destructor.

9.34.3 Dokumentation der Elementfunktionen

9.34.3.1 unsigned int getNumEntries ( )

Returns the size of the intern **TimeSpan** (S. 115) vector.

This function is deprecated. Use the std::vector::size function instead.

9.34.3.2 TimeSpan getEntry (unsigned int *index*)

Returns the *timeSpan* at *index*.

This function is deprecated. Use the std::vector::operator[] or the std::vector::at() function instead

Wird benutzt von TimeSpanContainer::appendTimeSpanContainer().

9.34.3.3 void appendTimeSpan (TimeSpan *timeSpan*)

Appends a time span to the container.

This function inserts a time span in the container at a position such that the time spans are sorted by their start time.

Benutzt TimeSpan::getEndTime() und TimeSpan::getStartTime().

Wird benutzt von TimeSpanContainer::appendTimeSpanContainer() und EventContainer::calculate-TimeSpans().

9.34.3.4 void appendTimeSpanContainer (TimeSpanContainer *container*)

Appends a time span container.

Benutzt TimeSpanContainer::appendTimeSpan() und TimeSpanContainer::getEntry().

Wird benutzt von EventContainer::calculateTimeSpans().

9.34.3.5 void mergeTimeSpansWithIntersection (bool *logMerging* = `true`)

Merges time spans that overlap each other.

This function merges overlapping time spans into single time spans.

Wird benutzt von EventContainer::calculateTimeSpans().

9.34.3.6 TimeSpan getIncludedTimeSpan (TimeSpan *timespan*)

Returns the time span that is included in the passed one.

If more than one time span is included in the passed *timespan* a time span with the start time of the first time span and the end time of the last time span is returned.

Benutzt TimeSpan::getStartTime().

9.34.3.7 void deleteAllEntries ( )

Deletes all entries.

This function is deprecated. use std::vector::clear instead.

9.34.3.8 string toString ( )

Debug function - converts class data to string.

9.35 TimeZone Klassenreferenz

Klassen

- struct **RegTimezoneInformation**

Öffentliche Methoden

- **TimeZone** ()
  Constructor.
- ~**TimeZone** ()
  Destructor.
- std::string **getTZ** (bool summertime)
- std::string **getDescription** ()
  Returns the description of the time zone (e.g. '(GMT) Dublin, Edinburgh, Lissabon, London').
- std::string **getKeyName** ()
  Returns the key name of the time zone (e.g. 'GMT_Standard_Time').
- std::string **getDlt** ()
  Returns the time zone's dlt name.
- int **getOffset** ()
  Returns the time zone offset to GMT in minutes.
- void **initFromRegistry** (std::string keyName, std::string description, std::string dlt, **RegTimezoneInformation** timeZoneInfo)

9.35.1 Beschreibung der Konstruktoren und Destruktoren

9.35.1.1 TimeZone ( )

Constructor.

9.35.1.2 ~TimeZone ( )

Destructor.

9.35.2 Dokumentation der Elementfunktionen

9.35.2.1 std::string getTZ (bool *summertime*)

9.35.2.2 std::string getDescription ( )

Returns the description of the time zone (e.g. '(GMT) Dublin, Edinburgh, Lissabon, London').

9.35.2.3 std::string getKeyName ( )

Returns the key name of the time zone (e.g. 'GMT_Standard_Time').

9.35.2.4 std::string getDlt ( )

Returns the time zone's dlt name.

9.35.2.5 int getOffset ( )

Returns the time zone offset to GMT in minutes.

9.35.2.6 void initFromRegistry (std::string *keyName,* std::string *description,* std::string *dlt,* RegTimezoneInformation *timeZoneInfo*)

9.36 TransferApplicationException Klassenreferenz

Class for transfer application errors, warnings, info.

Öffentliche Typen

- enum **TransferExceptionType** { **TRANS_ERROR**, **TRANS_CANCEL**, **TRANS_INFO**, **TRANS_WARNING** }

Öffentliche Methoden

- **TransferApplicationException** (std::string message, **TransferExceptionType** type)
  Creates a new exception with a specified message.
- std::string **getMessage** ()
  Returns the error message of the exception.
- **TransferExceptionType getType** ()
  Returns the exception type.
- uint32_t **getLastError** ()
  returns the last Windows system error

9.36.1 Ausführliche Beschreibung

Class for transfer application errors, warnings, info.

The exception handling of the trace data transfer is managed with this class. An **TransferApplicationException** (S. 121) can be from different types:

- error
- canceled
- info
- warning

Exceptions from type info and warning are only used internally - these types of notification are implemented by the **TransferApplicationListener** (S. 122) instead. Each exception contains an error message and the Windows specific last error (as returned by GetLastError()) to be able to debug errors related to e.g. file system failures.

9.36.2 Dokumentation der Aufzählungstypen

9.36.2.1 enum TransferExceptionType

Aufzählungswerte:

TRANS_ERROR
TRANS_CANCEL
TRANS_INFO
TRANS_WARNING

9.36.3 Beschreibung der Konstruktoren und Destruktoren

9.36.3.1 TransferApplicationException (std::string *message,* TransferExceptionType *type*) [inline]

Creates a new exception with a specified message.

The last Windows system error is inquired by GetLastError() when creating the Exception instance.

9.36.4 Dokumentation der Elementfunktionen

9.36.4.1 std::string getMessage ( ) [inline]

Returns the error message of the exception.

Wird benutzt von BluePiratClient::appendEvent(), BluePiratClient::BluePiratClient(), BluePiratClient::connectLogger(), BluePiratClient::convertOfflineData(), BluePiratClient::deleteAllFilesOnLogger(), BluePiratClient::deleteData(), BluePiratClient::detectLogger(), BluePiratClient::downloadAndConvertData(), BluePiratClient::downloadData(), BluePiratClient::downloadLogFiles(), BluePiratClient::getCurrentLoggerTime(), BluePiratClient::getDataLoggerName(), BluePiratClient::initOfflineConversion(), BluePiratClient::initTransfer(), BluePiratClient::keepLoggerAlive(), BluePiratClient::readConfigurationFromLogger(), BluePiratClient::setLoggerTime(), BluePiratClient::setLoggerTimeToSystemTime(), BluePiratClient::setMarker(), BluePiratClient::unprotectAllFilesOnLogger() und BluePiratClient::writeConfigurationToLogger().

9.36.4.2 TransferExceptionType getType ( ) [inline]

Returns the exception type.

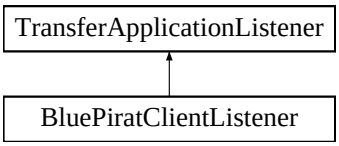
9.36.4.3 uint32_t getLastError ( ) [inline]

returns the last Windows system error

9.37 TransferApplicationListener Klassenreferenz

Class to listen the TransferApplication.

Klassendiagramm für TransferApplicationListener:



Öffentliche Methoden

- virtual void **onProgressDataDownload** (int percentCompleted, std::string status, unsigned int currentFileIndex, unsigned int totalNumberFiles)

  Called to indicate the current progress of a file transfer.
- virtual void **onProgressConversion** (int percentCompleted, std::string status)

  Called to indicate the current progress of file conversion.
- virtual void **onDownloadError** (std::vector< std::string > logVec)

  Called to indicate an error during file download.
- virtual void **onConversionError** (std::vector< std::string > logVec)

  Called to indicate an error during file conversion.
- virtual bool **isAbortRequested** ()

  Called to determine if the listener wants to abort the data transfer.
- virtual int **getProcessPrio** ()

  Called to determine the process priority the listener wants to set.
- virtual void **onApplicationInfo** (std::string infoMessage)

  Called to notify the listener about general information about the data transfer.
- virtual void **onApplicationWarning** (std::string warningMessage)

  Called to notify the listener about a warning.
- virtual void **showProgressDialog** (std::string, std::string title="", bool conversion=false)

  Called to display the progress dialog with the passed string.
- virtual void **setProgressDialogTitle** (std::string title)

  Sets the title of the progress dialog.
- virtual void **showBusyInfo** (std::string label)

  Called to inform the listener about a currently performed action.
- virtual void **deleteBusyInfo** ()

  Called to notify the listener about the end of a currently performed action.
- virtual **OverwritingResponse getOverwritingPermission** (std::string fileName)

  Called to get the permission from the user to overwrite a file.
- virtual bool **onWrongVersion** ()

  Called in case of wrong tmt file version.
- virtual bool **continueOnInsufficientDiskSpace** (std::string message)

  Called in case of not enough free diskspace.
- virtual int **onGetUserPassword** (std::string &pw, bool secondTimeOrMore)

  Called when the connection to the data logger failed due to a wrong password.
- virtual std::string **onCreateConvertedFileName** (std::string testerName, std::string loggerName, struct tm startTime, struct tm endTime, std::string channelName, **FormatId** format, std::vector< unsigned int > markerVector)

  Called to let the listener create the final name of a converted file.

- virtual bool **continueCanceledDownload** (std::string path)

  Called when a the client notifies that a requested offline data download was already started and canceled before.
- virtual void **onBlockDownload** (std::string path, std::string name)

  Called when a data block (trace, video, audio) was successfully downloaded.
- virtual void **onOfflineZIPRename** (std::string fileName)

  Called to notify about a splitted offline ZIP.
- virtual bool **onCorruptVideo** (std::string filename, bool &doContinue, bool &continueAll, bool &abort)
- virtual void **onDeleteFailed** (std::vector< std::string > files)
- virtual bool **onProgressDataDeletion** (int percentage, std::string label)

  Called on data deletion to update the progress dialog.
- virtual void **onProgressEnd** ()

  Called at the end of a progress.
- virtual void **onDataModelChanged** ()

  Called when the event container was updated.

9.37.1 Ausführliche Beschreibung

Class to listen the TransferApplication.

To ensure that another application, like the user interface for example, is able to react to events of the data transfer and conversion process, it must be inherited from **TransferApplicationListener** (S. 122). This class declares only virtual functions that must be defined in the inherited class.

9.37.2 Dokumentation der Elementfunktionen

9.37.2.1 virtual void onProgressDataDownload (int *percentCompleted,* std::string *status,* unsigned int *currentFileIndex,* unsigned int *totalNumberFiles*) [inline, virtual]

Called to indicate the current progress of a file transfer.

This function notifies the listener about the download progress of the raw Telemotive trace data. If the *percentCompleted* value has changed, but the *status* is still the same, the application passes an empty string as status to the function.

Parameter

percent-Completed	Percent of the entire download process (from 0...100%)
status	Status of the download process (e.g. "Downloading trace data. Block 5 of 32")
currentFile-Index	Index of the currently downloaded file
totalNumber-Files	Total number of the files to be downloaded

9.37.2.2 virtual void onProgressConversion (int *percentCompleted,* std::string *status*) [inline, virtual]

Called to indicate the current progress of file conversion.

This function notifies the listener about the conversion progress of the raw Telemotive trace data. If the *percentCompleted* value has changed, but the *status* is still the same, the application passes an empty string as status to the function.

Parameter

percent-Completed	Percent of the entire conversion process (from 0...100%)
status	Status of the conversion process (e.g. "Converting trace data. Block 5 of 32")

9.37.2.3 virtual void onDownloadError (std::vector< std::string > *logVec*) [inline, virtual]

Called to indicate an error during file download.

This function notifies the listener about download error that don't lead to an abort of the process. If one or more errors occur during download, the error messages are stored in a vector of string and passed to this function at the end of the entire download process.

9.37.2.4 virtual void onConversionError (std::vector< std::string > *logVec*) [inline, virtual]

Called to indicate an error during file conversion.

This function notifies the listener about conversion error that don't lead to an abort of the process. If one or more errors occur during conversion, the error messages are stored in a vector of string and passed to this function at the end of the entire conversion process.

9.37.2.5 virtual bool isAbortRequested ( ) [inline, virtual]

Called to determine if the listener wants to abort the data transfer.

This method should return true if the listener wants to abort the data transfer.

9.37.2.6 virtual int getProcessPrio ( ) [inline, virtual]

Called to determine the process priority the listener wants to set.

This method is called periodically and can be used to change the download and conversion process' priority. The listener can return one of the following values: 0 = THREAD_PRIORITY_IDLE 1 = THREAD_PRIORITY_NORMAL 2 = THREAD_PRIORITY_ABOVE_NORMAL

9.37.2.7 virtual void onApplicationInfo (std::string *infoMessage*) [inline, virtual]

Called to notify the listener about general information about the data transfer.

This method is used only to inform the listener about an issue without the listener being required to react in any way (as it would be in case of an application error). For example, this method is

called with an appropriate message when the user want to start the data transfer without selecting any data.

9.37.2.8 virtual void onApplicationWarning (std::string *warningMessage*) [inline, virtual]

Called to notify the listener about a warning.

Same as **onApplicationInfo()** (S. 125). Only separated in an extra function to be able to differ the icons when implementing a pop-up dialog to show the message.

Siehe auch

onApplicationInfo() (S. 125)

9.37.2.9 virtual void showProgressDialog (std::string , std::string *title* = "", bool *conversion* = false) [inline, virtual]

Called to display the progress dialog with the passed string.

This function should initialise a potential progress dialog.

Parameter

title	The title that should be displayed in the title bar of the dialog

9.37.2.10 virtual void setProgressDialogTitle (std::string *title*) [inline, virtual]

Sets the title of the progress dialog.

With this function the title of the progress dialog is changed without a new initialization of the entire dialog. E.g. switch between "data download" and "data conversion".

Parameter

title	The title that should be displayed in the title bar of the dialog

9.37.2.11 virtual void showBusyInfo (std::string *label*) [inline, virtual]

Called to inform the listener about a currently performed action.

Informs the listener about a currently performed action that has no associated process percentage. All process information besides the data download progress and the data conversion progress is notified through this function (e.g. "read data logger configuration" during the connection process). The end of the process action is either notified by **deleteBusyInfo()** (S. 127) or by another call of **showBusyInfo()** (S. 126).

Siehe auch

deleteBusyInfo() (S. 127)

9.37.2.12 virtual void deleteBusyInfo ( ) [inline, virtual]

Called to notify the listener about the end of a currently performed action.

Called to indicate the end of a currently performed action as described in **showBusyInfo()** (S. 126).

Siehe auch

showBusyInfo() (S. 126)

9.37.2.13 virtual OverwritingResponse getOverwritingPermission (std::string *fileName*) [inline, virtual]

Called to get the permission from the user to overwrite a file.

If the application wants to write an output trace file with a name that already exists, the listener has to give the overwriting permission. The default return value is 'NO'. If this function is not implemented in the listener class files can not be overwritten and the current output trace file is discarded.

Benutzt bp::NO.

9.37.2.14 virtual bool onWrongVersion ( ) [inline, virtual]

Called in case of wrong tmt file version.

This notifies the listener if the raw trace data format on the data logger is in a newer format than the client can handle. If this function returns true the conversion is continued anyway and unknown trace messages can not be converted.

9.37.2.15 virtual bool continueOnInsufficientDiskSpace (std::string *message*) [inline, virtual]

Called in case of not enough free diskspace.

This notifies the listener about not enough free disk space for data download or conversion. The user can continue or abort the process. Returning false will abort the process.

9.37.2.16 virtual int onGetUserPassword (std::string & *pw,* bool *secondTimeOrMore*) [inline, virtual]

Called when the connection to the data logger failed due to a wrong password.

This notifies the listener about a wrong entered password. Since the data logger is able to protect the trace files with a user password, this function is called if the usage of the default password doesn't work. The implementation of this function should return zero if a password was set to *pw*. To cancel the connection establishment return something greater than zero. The second parameter *secondTimeOrMore* indicates only whether this function is called the first time. With this one can distinguish between the user prompt, e.g. "enter password" for the first time, "wrong password, please try again" for the following times.

9.37.2.17 virtual std::string onCreateConvertedFileName (std::string *testerName,* std::string *loggerName,* struct tm *startTime,* struct tm *endTime,* std::string *channelName,* FormatId *format,* std::vector< unsigned int > *markerVector*) [inline, virtual]

Called to let the listener create the final name of a converted file.

The function passes several parameter to the listener that can be used to create the file name:

Parameter

testerName,:	Name of the tester from the client configuration
loggerName,:	Name of the logger from the logger configuration
startTime,:	Local start time of the trace file with the logger's time zone
endTime,:	Local end time of the trace file with the logger's time zone
channel-Name,:	Name of the included channel
format,:	Format of the trace file, see FormatId
marker-Vector,:	vector with the indices of all included markers

The parameter *channelName* is the channel's name from the logger configuration. If the trace file contains more than one trace channel, the channel name is created as follows:

- MOST control and asyncronous channel: MOST
- Multiple CAN channels: CAN
- Multiple SERIAL channels: SERIAL
- Multiple LIN channels: LIN
- Multiple channels of different type: MULTI

9.37.2.18 virtual bool continueCanceledDownload (std::string *path*) [inline, virtual]

Called when a the client notifies that a requested offline data download was already started and canceled before.

This function asks the listener whether to continue the canceled download or to start a new one

9.37.2.19 virtual void onBlockDownload (std::string *path,* std::string *name*) [inline, virtual]

Called when a data block (trace, video, audio) was successfully downloaded.

The passed string is the absolute path to the downloaded file

9.37.2.20 virtual void onOfflineZIPRename (std::string *fileName*) [inline, virtual]

Called to notify about a splitted offline ZIP.

When downloading offline data as ZIP archive, the application creates ZIPs up to a maximum size of 3,8 GB. This has to reasons:

1. FAT file systems support only files up to 4 GB

2. ZIP files greater than 4 GB on other file systems causes several problems depending on the used zip tool - included the blue PiraT client itself.

Including a buffer size, the application splits ZIP files at 3,8 GB. This function is called to notify the calling application about such a file splitting. The passed parameter *fileName* includes the file name that is used for the closed part of the ZIP archive.

9.37.2.21 virtual bool onCorruptVideo (std::string *filename,* bool & *doContinue,* bool & *continueAll,* bool & *abort*) [inline, virtual]

9.37.2.22 virtual void onDeleteFailed (std::vector< std::string > *files*) [inline, virtual]

Called when the deletion of one or more files failed

9.37.2.23 virtual bool onProgressDataDeletion (int *percentage,* std::string *label*) [inline, virtual]

Called on data deletion to update the progress dialog.

9.37.2.24 virtual void onProgressEnd ( ) [inline, virtual]

Called at the end of a progress.

9.37.2.25 virtual void onDataModelChanged ( ) [inline, virtual]

Called when the event container was updated.

9.38 VersionsInfoEntry Strukturreferenz

Öffentliche Attribute

- std::string **d_keyword**
- std::string **d_version**

9.38.1 Dokumentation der Datenelemente

9.38.1.1 std::string d_keyword

9.38.1.2 std::string d_version

Kapitel 10

Datei-Dokumentation

10.1 BluePiratClient.cc-Dateireferenz

10.2 BluePiratClientCommon.hh-Dateireferenz

Klassen

- class **BluePiratClient**
  Interface Class for the blue PiraT Client library.

Namensbereiche

- namespace **bp**

Variablen

- static const char * **BPC_LIB_VERSION** = "4.3.3"
  Version of the blue PiraT client library.

10.2.1 Variablen-Dokumentation

10.2.1.1 const char* BPC_LIB_VERSION = "4.3.3" [static]

Version of the blue PiraT client library.

Wird benutzt von BluePiratClient::BluePiratClient() und BluePiratClient::getLibVersion().

10.3 BluePiratClientException.hh-Dateireferenz

Declares the class BluePiratClientException.

Klassen

- class **BluePiratClientException**
  Class for blue PiraT Client library errors.

Namensbereiche

- namespace **bp**

10.3.1 Ausführliche Beschreibung

Declares the class BluePiratClientException.

Autor

Markus van Pinxteren

Datum

29.10.2007

10.4 BluePiratClientListener.hh-Dateireferenz

Class to listen the Blue PiraT Client Library.

Klassen

- class **BluePiratClientListener**

Namensbereiche

- namespace **bp**

10.4.1 Ausführliche Beschreibung

Class to listen the Blue PiraT Client Library.

Autor

Markus van Pinxteren

Datum

30.10.2007

10.5 CanFilterProperties.cc-Dateireferenz

This class stores CAN filters.

10.5.1 Ausführliche Beschreibung

This class stores CAN filters.

Autor

Muhammad, Sohail

Datum

10.6 CanFilterProperties.hh-Dateireferenz

This property class stores CAN filters.

Klassen

- class **CanFilterProperties**
  Property class to store the filtered CAN IDs.

Namensbereiche

- namespace **bp**

10.6.1 Ausführliche Beschreibung

This property class stores CAN filters.

Autor

Muhammad, Sohail

Datum

10.7 CanProperties.cc-Dateireferenz

Property class to store the CAN properties of one channel.

10.7.1 Ausführliche Beschreibung

Property class to store the CAN properties of one channel.

Autor

Muhammad, Sohail

Datum

10.8 CanProperties.hh-Dateireferenz

Property class to store the CAN properties of a CAN interface.

Klassen

- class **CanProperties**
  Property class to store the CAN properties of a CAN interface.

Namensbereiche

- namespace **bp**

10.8.1 Ausführliche Beschreibung

Property class to store the CAN properties of a CAN interface.

Autor

Muhammad, Sohail

Datum

10.9 CascadingProperties.cc-Dateireferenz

Class to store the cascading properties.

10.9.1 Ausführliche Beschreibung

Class to store the cascading properties.

Autor

Muhammad, Sohail

Datum

28.04.2006

10.10 CascadingProperties.hh-Dateireferenz

Class to store the cascading properties.

Klassen

- class **CascadingProperties**
  This class stores the properties for the logger's cascading feature.

Namensbereiche

- namespace **bp**

10.10.1 Ausführliche Beschreibung

Class to store the cascading properties.

Autor

Muhammad, Sohail

Datum

28.04.2006

10.11 Channel.hh-Dateireferenz

Class to represent one channel.

Klassen

- class **Channel**
  Class to represent one channel.

Namensbereiche

- namespace **bp**

10.11.1 Ausführliche Beschreibung

Class to represent one channel.

Autor

Markus van Pinxteren

Datum

07.11.2005

10.12 ClientConfiguration.cc-Dateireferenz

Class to configure the client library settings.

Funktionen

- **__declspec** (deprecated) void **ClientConfiguration**

10.12.1 Ausführliche Beschreibung

Class to configure the client library settings.

Autor

Markus van Pinxteren

Datum

24.08.2007

10.12.2 Dokumentation der Funktionen

10.12.2.1 __declspec (deprecated)

When selecting a marker event for transfer, the data of a user defined time span around the marker is transferred. This time span can be defined by this function, setting the pre- and the post-marker time. By default (without calling loadSettings()) pre- and posttime are both 20 seconds. Passing a negative value to those parameters will set the start/end time of the time span to the last startup respectively the next shutdown before/after the marker time. If you want to download the trace data of marker events to an offline data set, you have to specify this settings and pass the configuration to the BluePiratClient instance before calling downloadData(). Otherwise the default values are used.

Parameter

bus	The channel bus type for that the marker time span should be set (see **TypeDefs.hh** (S. 156) for all possible ChannelTypeIds)
preTime	Seconds to transfer before the marker event (-1 for last startup)
postTime	Seconds to transfer after the marker event (-1 for next shutdown)

Siehe auch

loadSettings(), setClientConfiguration()

Benutzt bp::LAST_STARTUP, bp::NEXT_SHUTDOWN und Channel::toString().

10.13 ClientConfiguration.hh-Dateireferenz

Class to configure the client library settings.

Klassen

- class **ClientConfiguration**
  Class to configure the client library settings.

Namensbereiche

- namespace **bp**

10.13.1 Ausführliche Beschreibung

Class to configure the client library settings.

Autor

Markus van Pinxteren

Datum

24.08.2007

10.14 DataStorageProperties.cc-Dateireferenz

The DataStorageProperties stores settings that consider the storage and protection of recorded trace data.

10.14.1 Ausführliche Beschreibung

The DataStorageProperties stores settings that consider the storage and protection of recorded trace data.

Autor

Muhammad, Sohail

Datum

10.15 DataStorageProperties.hh-Dateireferenz

The property class stores settings that consider the storage and protection of recorded trace data.

Klassen

- class **DataStorageProperties**
  The property class stores settings that consider the storage and protection of recorded trace data.

Namensbereiche

- namespace **bp**

10.15.1 Ausführliche Beschreibung

The property class stores settings that consider the storage and protection of recorded trace data.

Autor

Muhammad, Sohail

Datum

10.16 EventContainer.cc-Dateireferenz

Class to handle the Events file and its entries.

10.16.1 Ausführliche Beschreibung

Class to handle the Events file and its entries.

Autor

Markus van Pinxteren

Datum

10.01.2005

Siehe auch

EventContainerEntry

10.17 EventContainer.hh-Dateireferenz

Class to handle the Events file and its entries.

Klassen

- class **EventContainerListener**
- class **EventContainer**

  Class to store the entries of the event file.

Namensbereiche

- namespace **bp**

10.17.1 Ausführliche Beschreibung

Class to handle the Events file and its entries.

Autor

Markus van Pinxteren

Datum

10.01.2005

Siehe auch

EventContainerEntry

10.18 EventContainerEntry.cc-Dateireferenz

Class to encapsulate one Event file entry.

10.18.1 Ausführliche Beschreibung

Class to encapsulate one Event file entry.

Autor

Markus van Pinxteren

Datum

17.01.2005

Siehe auch

EventContainer

10.19 EventContainerEntry.hh-Dateireferenz

Class to encapsulate one Event file entry.

Klassen

- class **EventContainerEntry**
  Class to encapsulate one Events file entry.

Namensbereiche

- namespace **bp**

10.19.1 Ausführliche Beschreibung

Class to encapsulate one Event file entry.

Autor

Markus van Pinxteren

Datum

17.01.2005

Siehe auch

EventContainer

10.20 FlexrayGeneralProperties.cc-Dateireferenz

This class enables to read/write flexray general property from/to the ConfigConatiner.

10.20.1 Ausführliche Beschreibung

This class enables to read/write flexray general property from/to the ConfigConatiner.

Autor

Muhammad, Sohail

Datum

16.10.2007

10.21 FlexrayGeneralProperties.hh-Dateireferenz

This class enables to read/write flexray general properties from/to the ConfigConatiner.

Klassen

- class **FlexrayGeneralProperties**
  This class enables to read/write flexray general properries from/to the ConfigConatiner.

10.21.1 Ausführliche Beschreibung

This class enables to read/write flexray general properties from/to the ConfigConatiner.

Autor

Muhammad, Sohail

Datum

16.10.2007

10.22 FlexrayProperties.cc-Dateireferenz

This class enables to read/write flexray property from/to the ConfigConatiner.

10.22.1 Ausführliche Beschreibung

This class enables to read/write flexray property from/to the ConfigConatiner.

Autor

Muhammad, Sohail

Datum

16.10.2007

10.23 FlexrayProperties.hh-Dateireferenz

This class enables to read/write flexray properties from/to the ConfigConatiner.

Klassen

- class **FlexrayProperties**
  This class enables to read/write flexray properries from/to the ConfigConatiner.

10.23.1 Ausführliche Beschreibung

This class enables to read/write flexray properties from/to the ConfigConatiner.

Autor

Muhammad, Sohail

Datum

16.10.2007

10.24 FormatId.hh-Dateireferenz

Possible formats the trace data can be converted to.

Aufzählungen

- enum **FormatId** {
  NOTRANSFER = 0, **TMASC** = 1, **OP2** = 2, **CANOE** = 3,
  STA = 4, **GNLOG_SINGLE** = 5, **TCLOG** = 6, **TCLOG_TS** = 7,
  RAW_SERIAL = 8, **IMG** = 10, **TM** = 11, **CANCORDER** = 12,
  GNLOG_SHARED = 13, **MPEG4_BLOCKS** = 14, **APN** = 15, **ASCHEX** = 16,
  IPOD_COMMAND = 17, **WAV** = 19, **TCPDUMP** = 21, **MPEG4_JOINED_HQ** = 22,
  BLF = 23, **MDF** = 24, **DLT_BMW** = 25, **ETH_RAW** = 26,
  INVALID = 0xFF }

10.24.1 Ausführliche Beschreibung

Possible formats the trace data can be converted to.

Autor

Markus van Pinxteren

Datum

17.01.2006

10.25 GeneralProperties.cc-Dateireferenz

This class enables to read/write general property from/to the ConfigConatiner.

10.25.1 Ausführliche Beschreibung

This class enables to read/write general property from/to the ConfigConatiner.

Autor

Muhammad, Sohail

Datum

10.26 GeneralProperties.hh-Dateireferenz

Property class to store general logger settings.

Klassen

- class **GeneralProperties**
  Property class to store general logger settings.

Namensbereiche

- namespace **bp**

10.26.1 Ausführliche Beschreibung

Property class to store general logger settings.

Autor

Muhammad, Sohail

Datum

10.27 LinProperties.cc-Dateireferenz

This class enables to read/write lin property from/to the ConfigConatiner.

10.27.1 Ausführliche Beschreibung

This class enables to read/write lin property from/to the ConfigConatiner.

Autor

Muhammad, Sohail

Datum

10.04.2007

10.28 LinProperties.hh-Dateireferenz

Property class to store the LIN settings.

Klassen

- class **LinProperties**
  Property class to store the LIN settings.

Namensbereiche

- namespace **bp**

10.28.1 Ausführliche Beschreibung

Property class to store the LIN settings.

Autor

Muhammad, Sohail

Datum

10.04.2007

10.29 LoggerConfiguration.cc-Dateireferenz

Class to configure the data logger via client library.

10.29.1 Ausführliche Beschreibung

Class to configure the data logger via client library.

Autor

Markus van Pinxteren

Datum

10.10.2007

10.30 LoggerConfiguration.hh-Dateireferenz

Class to configure the data logger via client library.

Klassen

- class **LoggerConfiguration**
  Class to configure the data logger via client library.

Namensbereiche

- namespace **bp**

10.30.1 Ausführliche Beschreibung

Class to configure the data logger via client library.

Autor

Markus van Pinxteren

Datum

09.10.2007

10.31 LoggerDetectorListener.hh-Dateireferenz

Listener class.

Klassen

- class **LoggerDetectorListener**
  Base class to listen the LoggerDetector class.

10.31.1 Ausführliche Beschreibung

Listener class.

Autor

Markus van Pinxteren

Datum

13.12.2007

10.32 mainpage.txt-Dateireferenz

10.33 MarkerProperties.cc-Dateireferenz

This class enables to read/write marker properties from/to the ConfigConatiner.

10.33.1 Ausführliche Beschreibung

This class enables to read/write marker properties from/to the ConfigConatiner.

Autor

Muhammad, Sohail

Datum

10.34 MarkerProperties.hh-Dateireferenz

Property class to store the CAN messages related to the markers.

Klassen

- class **MarkerProperties**
  Property class to store the CAN messages related to the markers.

Namensbereiche

- namespace **bp**

10.34.1 Ausführliche Beschreibung

Property class to store the CAN messages related to the markers.

Autor

Muhammad, Sohail

Datum

10.35 MarkerProtectionProperties.cc-Dateireferenz

This class enables to read/write buffer protected marker properties from/to the ConfigConatiner.

10.35.1 Ausführliche Beschreibung

This class enables to read/write buffer protected marker properties from/to the ConfigConatiner.

Autor

Muhammad, Sohail

Datum

10.36 MarkerProtectionProperties.hh-Dateireferenz

Property class to store the length of the data block to protect around a marker time stamp.

Klassen

- class **MarkerProtectionProperties**
  Property class to store the length of the data block to protect around a marker time stamp.

Namensbereiche

- namespace **bp**

10.36.1 Ausführliche Beschreibung

Property class to store the length of the data block to protect around a marker time stamp.

Autor

Muhammad, Sohail

Datum

10.37 MostFilterProperties.cc-Dateireferenz

This class enables to read/write MOST filter properties from/to the ConfigConatiner.

Makrodefinitionen

- #define **dontCare** -1

10.37.1 Ausführliche Beschreibung

This class enables to read/write MOST filter properties from/to the ConfigConatiner.

Autor

Muhammad, Sohail

Datum

10.37.2 Makro-Dokumentation

10.37.2.1 #define dontCare -1

10.38 MostFilterProperties.hh-Dateireferenz

Property class to store most filter.

Klassen

- class **MostFilterProperties**
  Property class to store most filter.
- struct **MostFilterProperties::MostFilter**
  Strcut that stores a set of MOST parameters.

Namensbereiche

- namespace **bp**

10.38.1 Ausführliche Beschreibung

Property class to store most filter.

Autor

Muhammad, Sohail

Datum

10.39 MostProperties.cc-Dateireferenz

Property class to store the MOST configuration settings.

10.39.1 Ausführliche Beschreibung

Property class to store the MOST configuration settings.

Autor

Muhammad, Sohail

Datum

28.04.2006

10.40 MostProperties.hh-Dateireferenz

Property class to store the MOST configuration settings.

Klassen

- class **MostProperties**
  Property class to store the MOST configuratio settings.

Namensbereiche

- namespace **bp**

10.40.1 Ausführliche Beschreibung

Property class to store the MOST configuration settings.

Autor

Muhammad, Sohail

Datum

28.04.2006

10.41 NetworkConfigProperties.cc-Dateireferenz

This class enables to read / write dhcp config properties from/to the ConfigConatiner.

10.41.1 Ausführliche Beschreibung

This class enables to read / write dhcp config properties from/to the ConfigConatiner.

Autor

Muhammad, Sohail

Datum

10.42 NetworkConfigProperties.hh-Dateireferenz

Property class to store the serial interface settings.

Klassen

- class **NetworkConfigProperties**
  Property class to store the dhcp settings.

Namensbereiche

- namespace **bp**

10.42.1 Ausführliche Beschreibung

Property class to store the serial interface settings.

Autor

Muhammad, Sohail

Datum

10.43 RCVoiceProperties.cc-Dateireferenz

Property class to store the RCVoice configuration settings.

10.43.1 Ausführliche Beschreibung

Property class to store the RCVoice configuration settings.

Autor

Muhammad, Sohail

Datum

31.03.2008

10.44 RCVoiceProperties.hh-Dateireferenz

Property class to store the RCVoice configuration settings.

Klassen

- class **RCVoiceProperties**
  Property class to store the RCVoice configuration settings.

Namensbereiche

- namespace **bp**

10.44.1 Ausführliche Beschreibung

Property class to store the RCVoice configuration settings.

Autor

Muhammad, Sohail

Datum

31.03.2008

10.45 SerialProperties.cc-Dateireferenz

This class enables to read / write serial properties from/to the ConfigConatiner.

10.45.1 Ausführliche Beschreibung

This class enables to read / write serial properties from/to the ConfigConatiner.

Autor

Muhammad, Sohail

Datum

10.46 SerialProperties.hh-Dateireferenz

Property class to store the serial interface settings.

Klassen

- class **SerialProperties**
  Property class to store the serial interface settings.

Namensbereiche

- namespace **bp**

10.46.1 Ausführliche Beschreibung

Property class to store the serial interface settings.

Autor

Muhammad, Sohail

Datum

10.47 SleepProperties.cc-Dateireferenz

Property class to store the power management settings.

10.47.1 Ausführliche Beschreibung

Property class to store the power management settings.

Autor

Muhammad, Sohail

Datum

10.48 SleepProperties.hh-Dateireferenz

Property class to store the power management settings.

Klassen

- class **SleepProperties**
  Property class to store the power management settings.

Namensbereiche

- namespace **bp**

10.48.1 Ausführliche Beschreibung

Property class to store the power management settings.

Autor

Muhammad, Sohail

Datum

10.49 TimeSpan.cc-Dateireferenz

Class to store a time span in usec.

10.49.1 Ausführliche Beschreibung

Class to store a time span in usec.

Autor

Markus van Pinxteren

Datum

17.01.2005

10.50 TimeSpan.hh-Dateireferenz

Class to store a time span in usec.

Klassen

- class **TimeSpan**
  Class to store a time span.

Namensbereiche

- namespace **bp**

10.50.1 Ausführliche Beschreibung

Class to store a time span in usec.

Autor

Markus van Pinxteren

Datum

17.01.2005

10.51 TimeSpanContainer.cc-Dateireferenz

Class to store a number of TimeSpans.

10.51.1 Ausführliche Beschreibung

Class to store a number of TimeSpans.

Autor

Markus van Pinxteren

Datum

10.01.2005

Siehe auch

TimeSpan

10.52 TimeSpanContainer.hh-Dateireferenz

Class to store a number of TimeSpans.

Klassen

- class **TimeSpanContainer**
  Class to store a number of TimeSpans.

Namensbereiche

- namespace **bp**

10.52.1 Ausführliche Beschreibung

Class to store a number of TimeSpans.

Autor

Markus van Pinxteren

Datum

10.01.2005

Siehe auch

TimeSpan

10.53 TimeZone.hh-Dateireferenz

Class that represent the.

Klassen

- class **TimeZone**
- struct **TimeZone::RegTimezoneInformation**

Namensbereiche

- namespace **bp**

10.53.1 Ausführliche Beschreibung

Class that represent the.

Autor

Muhammad Sohail

Datum

24.04.2007

10.54 TransferApplicationException.hh-Dateireferenz

Declares the class TransferApplicationException.

Klassen

- class **TransferApplicationException**
  Class for transfer application errors, warnings, info.

Namensbereiche

- namespace **bp**

10.54.1 Ausführliche Beschreibung

Declares the class TransferApplicationException.

Autor

Markus van Pinxteren

Datum

04.04.2005

10.55 TransferApplicationListener.hh-Dateireferenz

Class to listen the trace file transfer and conversion process.

Klassen

- class **TransferApplicationListener**
  Class to listen the TransferApplication.

Namensbereiche

- namespace **bp**

10.55.1 Ausführliche Beschreibung

Class to listen the trace file transfer and conversion process.

Autor

Markus van Pinxteren

Datum

10.01.2005

10.56 TypeDefs.hh-Dateireferenz

Type definitions for blue PiraT Client Library.

Klassen

- struct **LoggerInNetwork**
- struct **VersionsInfoEntry**
- struct **CANChannelClockFrequency**

  Type definition for the Configuration Tool.
- struct **ChangedDBPathChannel**
- struct **InterfaceParametersStruct**

  Struct for interface parameters.

Namensbereiche

- namespace **bp**

Aufzählungen

- enum **EventType** {
  HEADLINE = 0, **STARTUP** = 1, **MARKER** = 2, **SHUTDOWN** = 3,
  DATAERASED = 4, **SLAVEOFFSET** = 5, **SLAVETOMASTER** = 6, **INFO** = 7,
  DATADOWNLOAD = 8, **ERROR_EVENT** = 9, **SETTIME** = 10, **NEWTIME** = 11,
  UNKNOWN_EVENT = 99 }

  Event types used by application and user interface, HEADLINE only for GUI.
- enum **OverwritingResponse** {
  YES = 0, **NO**, **YES_TO_ALL**, **NO_TO_ALL**,
  ABORT }
- enum **LoggerStatus** { **LOGGER_FOUND**, **LOGGER_NOT_FOUND** }

  Logger status.
- enum **CONFIG_IO_MODE** {
  CONFIG_WRITE_TO_LOGGER, **CONFIG_READ_FROM_LOGGER**, **CONFIG_SAVE_LOCALLY**, **CONFIG_LOAD_LOCALLY**,
  CONFIG_PASSWORD_UPDATE, **CONFIG_DATE_TIME_UPDATE** }

  Type definition for the Configuration Tool.
- enum **FirmwareFeatures** {
  NONE_FIRMWARE_FEATURE, **CAN_ACKNOWLEDGE_MODE**, **CAN_BIT_TIMING_MODE**, **LOGGER_CASCADING_FEATURE**,
  MOST_FEATURE, **MOST_TRACE_ASYNC_CHANNEL_MODE**, **MOST_ASYNC_ARBITRATION_VALUE_MODE**, **MOST_DATA_REGENERATION_MODE**,
  RECORDING_MOST_ERROR_MESSAGES, **MOST_TRACE_SYNC_CHANNEL_MODE**,
  MOST50_TRACE_SYNC_CHANNEL_MODE, **LOGGER_AUTOMATIC_SHUTDOWN_DEACTIVATED_MODE**,
  GNLOG_OVER_ETHERNET_FEATURE, **ETHERNET_LOGGING_EXTENDED_IP_CONFIG**, **CAN_TRIGGER_MASKING_FEATURE**, **CAMERA_FEATURE**,
  LIN_FEATURE, **RC_MONITOR_FEATURE**, **COMPLEX_TRIGGER_FEATURE**, **CAMERA_EXTENTION_FEATURE**,
  MULTIPLE_CAMERA_SUPPORT_FEATURE, **TRACE_FILE_COMPRESSION_FEATUR-**

E, **EXTENDED_TRIGGER_CONFIG_FEATURE**, **TRIGGER_TYPE_CONFIG_FEATURE**, **FLEXRAY_FEATURE**, **ETHERNET_LOGGING_PORT**, **RC_VOICE_FEATURE**, **CASCADING_CHANNEL_OFFSET_EXTENSION**,
NETWORK_CONFIG_FEATURE, **MOST150_FEATURE**, **MOST150_FILTER_FEATURE**, **CAMERA_PASSWORD_FEATURE**,
ETHERNET_RAW_UTF8_LOGGING_FEATURE, **INTERFACE_NAME_MAPPING_FEATURE**, **ETHERNET_TIMEOUT_FEATURE**, **ETHERNET_UDP_SERVER_FEATURE**, **MOST50_FEATURE**, **MOST50_FILTER_FEATURE**, **DLT_BMW_LOGGING_FEATURE**, **ETHERNET_VLAN_FEATURE**,
MOST_TRAINING }

Type definition for the Configuration Tool.

- enum **LoggerFeatures** {
NONE_LOGGER_FEATURE, **NUM_CAN_CHANNELS**, **NUM_SW_CHANNELS**, **NUM_TRANSCEIVER_TYPES**,
ETHERNET_SWITCH, **NUM_MOST25_CHANNELS** }
- enum **FeaturesType** { **FIRMWARE_CAPABILITIES**, **DEVICE_CAPABILITIES**, **LICENSES**, **FLEXRAY_CONFIG** }

  Type definition for the Configuration Tool.
- enum **ConfigType** { **OCEAN_INI**, **TRIGGER_INI** }
- enum **LocalLanguage** { **GERMAN**, **US_ENGLISH** }
- enum **LowerBoundType** { **PRETIME**, **LAST_STARTUP**, **UNKNOWN_LB** }
- enum **UpperBoundType** {
POSTTIME, **NEXT_SHUTDOWN**, **NEXT_MARKER_INFO**, **NEXT_TEXT_INFO**, **UNKNOWN_UB** }

10.56.1 Ausführliche Beschreibung

Type definitions for blue PiraT Client Library.

Autor

Markus van Pinxteren

Datum

10.01.2006

Index