



blue PiraT 2 Client Library 1.7.7

User's manual

Generated by Doxygen 1.8.0

Fri Jan 25 2013 17:40:30

# Inhaltsverzeichnis

<b>1</b>	<b>User's manual - blue PiraT 2 Client Library 1.7.7</b>	<b>1</b>
1.1	General	1
1.2	Functionality	1
1.3	Compiler/Linker	2
1.4	Thread safety	2
1.5	Demo project	2
<b>2</b>	<b>Module Index</b>	<b>10</b>
2.1	Modules	10
<b>3</b>	<b>Class Index</b>	<b>11</b>
3.1	Class Hierarchy	11
<b>4</b>	<b>Class Index</b>	<b>12</b>
4.1	Class List	12
<b>5</b>	<b>File Index</b>	<b>14</b>
5.1	File List	14
<b>6</b>	<b>Module Documentation</b>	<b>15</b>
6.1	Download Functions	15
6.2	Conversion Functions	16
<b>7</b>	<b>Class Documentation</b>	<b>17</b>
7.1	BP2Device Struct Reference	17
7.2	BPNGError Struct Reference	17
7.3	BPNGLoggerDetector Class Reference	18
7.4	CANPseudoMessagesProperties Struct Reference	21
7.5	ClientProperties Struct Reference	23
7.6	CommonProperties Struct Reference	24
7.7	DataSpan Struct Reference	25
7.8	IBPNGClient Struct Reference	25
7.9	IBPNGClientListener Struct Reference	37
7.10	IChannel Struct Reference	41
7.11	IChannelList Struct Reference	41
7.12	IConversionSet Struct Reference	42
7.13	IFormatInfo Struct Reference	43
7.14	IFormatList Struct Reference	44
7.15	IRdbEvent Struct Reference	45
7.16	IRdbEventList Struct Reference	45
7.17	MOSTPseudoMessagesProperties Struct Reference	46
7.18	OnlineLoggerInfo Struct Reference	46
7.19	RdbEvent Struct Reference	47
7.20	RdbEventList Class Reference	48

<b>8</b>	<b>File Documentation</b>	<b>50</b>
8.1	BPNGDefines.h File Reference . . . . .	50
8.2	IBPNGClient.h File Reference . . . . .	57
8.3	IBPNGClientListener.h File Reference . . . . .	58

## Kapitel 1

# User's manual - blue PiraT 2 Client Library

### 1.7.7

#### 1.1 General

This is the documentation for the C++ blue PiraT 2 Client library which is compatible with all Microsoft compilers. The library's interface class [IBPNGClient](#) uses only base data type parameters like *int*, *long* and *char*, pointers to those types and pointers to complex proprietary data objects that are entirely defined within the library. To access the data of such objects the library comes with own interface definitions for all of those complex data types (like e.g. [IRdbEventList](#), see [BPNGDefines.h](#)). All library functions are blocking functions. Status and progress information is processed via listener callbacks (see [IBPNGClientListener](#)). Errors are processed by the functions' return values (see section Error handling for more details).

#### 1.2 Functionality

The blue PiraT 2 client library provides methods for base functionality like:

- downloading the logger's raw trace data as offline data sets
- converting trace data to nearly all common file formats
- reading and reconfiguring the data logger
- updating the logger's firmware
- creating bug reports

Besides that there are several more functions for deleting data, setting the logger's time and marker, scanning the network for available loggers, etc.

##### 1.2.1 Error handling and listener mechanism

All errors are processed by the functions' return values. If the return value states an error a call to `getLastError()` provides details about the error(s) occurred. Warnings are not intended to abort

a process. That's why they are reported via the function `IBPNGClientListener::onWarning()`. It's up to the user to handle them or not.

Progress and status information is also processed via listener callbacks. You have to derive your own class from `IBPNGClientListener` and implement all functions you need. Register an object of your listener class at the executing `IBPNGClient` with `IBPNGClient::addListener()`.

## 1.3 Compiler/Linker

The library is build with Microsoft Visual C++ and is linked to the C-Runtime Library with the Multi-threaded resp. Multi-threaded Debug compiler switch (/MT resp. /MTd). The user's project must have the same settings. Applications with mixed runtime library linkage may cause errors that are difficult to diagnose and to handle. The debug version of the library is named with a "\_d" suffix.

## 1.4 Thread safety

The library is thread safe when using different objects of `IBPNGClient` resp. the objects' pointers in different threads. It is NOT thread safe for one `IBPNGClient` instance in several threads!

## 1.5 Demo project

The "sample" directory contains a demo project for the blue PiraT 2 Client library.

Beispiel zur Verwendung der Library:

```
/*
    Sample project for BPNGClientLib (blue PiraT 2 Client Library)

    Link static runtime library (/MTd resp. /MT)
*/
#include "BPNGDefines.h"
#include "IBPNGClient.h"
#include "IBPNGClientListener.h"

#include "BPNGLoggerDetector.hh"
#include "RdbEventList.hh"

#include <iostream>
#include <direct.h>
#include <errno.h>
#include <time.h>

using namespace std;

void sampleFunctionDownload(BP2Device device);
void sampleFunctionOnlineConversion(BP2Device device);
void sampleFunctionOfflineConversion();
void sampleFunctionConfiguration(BP2Device device);

void main()
{
    // Get list of all currently available blue PiraT 2 devices
    BPNGLoggerDetector detector;
```

```

vector<BP2Device> devices = detector.getLoggerList(0);

if (devices.size() == 0)
    return; // no logger found

// select the device you want to work with
BP2Device device = devices[0];

sampleFunctionDownload(device);
//sampleFunctionOnlineConversion(device);
//sampleFunctionOfflineConversion();
//sampleFunctionConfiguration(device);
}

// We want to download all traces since last startup to an offline data set
void sampleFunctionDownload(BP2Device device)
{
    IBPNGClient* client = getBPNGClient();
    // connect logger
    BOOL ret = client->connectLogger(device.ipAddress.c_str());
    if (ret == 0)
    {
        BPNGError err = client->getLastError();
        cout << "Failed to connect logger. " << endl;
        cout << "BPNGErrCode: " << err.code << ", " << err.msg << endl;
        client->release();
        return;
    }

    ret = client->initOnline();
    if (ret == 0)
    {
        BPNGError err = client->getLastError();
        cout << "Failed to init online." << endl;
        cout << "BPNGErrCode: " << err.code << ", " << err.msg << endl;
        client->disconnectLogger();
        client->release();
        return;
    }

    IRdbEventList* list = client->getEventList();
    RdbEventList eventList(list);

    if (eventList.size() == 0)
    {
        cout << "Empty event list" << endl;
        client->disconnectLogger();
        client->release();
        return;
    }

    uint64_t startupId = 0;
    uint64_t endId = -1; //max value for uint64 to include everything in
    the id range

    // search last startup
    for (int i = eventList.size() - 1; i >= 0; --i)
    {
        if (eventList[i].type == EVID_STARTUP)
        {
            startupId = eventList[i].uniqueID;
            break;
        }
    }

    DataSpan span;
    span.type = DST_IDSPAN;

```

```

    span.start = startupId;
    span.end = endId;

    // if you want to download several spans, put them in a vector
    vector<DataSpan> spanVec;
    spanVec.push_back(span);

    ret = _mkdir("../testoutdir");
    if (ret != 0 && errno != EEXIST)
    {
        cout << "Failed to create output directory" << endl;
        client->disconnectLogger();
        client->release();
        return;
    }

    ret = client->downloadDataSpans(spanVec.size(), &spanVec[0], "../testoutdir\\BP2_Offline.zip", 0);
    if (ret == 0)
    {
        BPNGError err = client->getLastError();
        cout << "Failed to download data." << endl;
        cout << "BPNGErrorCode: " << err.code << ", " << err.msg << endl;
        client->disconnectLogger();
        client->release();
        return;
    }

    // disconnect
    client->disconnectLogger();
    // free memory
    client->release();
}

// We want to convert all CAN traces from the logger
// around the last Marker to CANoe asc and BLF format.
void sampleFunctionOnlineConversion(BP2Device device)
{
    IBPNGClient* client = getBPNGClient();

    // connect logger
    BOOL ret = client->connectLogger(device.ipAddress.c_str());
    if (ret == 0)
    {
        BPNGError err = client->getLastError();
        cout << "Failed to connect logger." << endl;
        cout << "BPNGErrorCode: " << err.code << ", " << err.msg << endl;
        client->release();
        return;
    }

    ret = client->initOnline();
    if (ret == 0)
    {
        BPNGError err = client->getLastError();
        cout << "Failed to init online." << endl;
        cout << "BPNGErrorCode: " << err.code << ", " << err.msg << endl;
        client->disconnectLogger();
        client->release();
        return;
    }

    IRdbEventList* list = client->getEventList();
    RdbEventList eventList(list);
    if (eventList.size() == 0)
    {
        cout << "Empty event list" << endl;
    }
}

```

```

        client->disconnectLogger();
        client->release();
        return;
    }

    uint64_t markerTimeStamp = 0;
    // search last marker
    for (int i = eventList.size() - 1; i >= 0; --i)
    {
        if (eventList[i].type == EVID_MARKER)
        {
            markerTimeStamp = eventList[i].timeStamp;
            break;
        }
    }

    if (markerTimeStamp == 0)
    {
        cout << "No marker found." << endl;
        client->disconnectLogger();
        client->release();
        return;
    }

    // Ensure the out directory exists
    ret = _mkdir("../testoutdir");
    if (ret != 0 && errno != EEXIST)
    {
        cout << "Failed to create output directory" << endl;
        client->disconnectLogger();
        client->release();
        return;
    }

    // Get a conversion set
    IConversionSet* conversionSet = client->createNewConversionSet();

    // The time span has to be 60s before and 60s after the marker
    uint64_t startTime = markerTimeStamp - 60 * 1000000; // in usec
    uint64_t endTime = markerTimeStamp + 60 * 1000000; // in usec

    // If you want to convert more than one span,
    // call this function several times
    conversionSet->addTimeSpan(startTime, endTime);

    // CAN #1 and CAN #2 are supposed to be written to one asc output file
    // each.
    // CAN #3 and CAN #4 are supposed to be written together in another asc
    // file.
    // All other CAN channels are supposed to be written together in one
    // BLF file.
    const IChannelList* channels = client->getLoggerChannels();
    for (int i = 0; i < channels->getSize(); ++i)
    {
        ChannelType type = channels->getChannel(i)->getType();
        if (type != CH_CANLS && type != CH_CANHS)
            continue;

        // Note: channel indices are zero based
        int index = channels->getChannel(i)->getIndex();
        if (index == 0 || index == 1)
        {
            // CAN #1 and #2 in separate files
            // -1 as fileId parameter creates a separate file for
            this channel
            conversionSet->addChannel(type, index, CANOE, -1);

```



```

    }
    else if (index == 2 || index == 3)
    {
        // CAN #3 and #4 in the same file.
        // fileId != -1 will write all channels with the same
format and same        // file Id to the same output file (if procurable in
accordance with        // the format specification.
        conversionSet->addChannel(type, index, CANOE, 10);
    }
    else
    {
        // All other CAN channels to one BLF file.
        conversionSet->addChannel(type, index, BLF, 20);
    }
}

ret = client->convertData(conversionSet, "..\\testoutdir");
if (ret == 0)
{
    BPNGError err = client->getLastError();
    cout << "Failed to convert data." << endl;
    cout << "BPNGErrCode: " << err.code << ", " << err.msg << endl;
    client->disconnectLogger();
    client->release();
    return;
}

// disconnect
client->disconnectLogger();
// free memory
client->release();
}

// We want to convert all CAN traces from an Offline data set
// around the last Marker to CANoe asc and BLF format.
void sampleFunctionOfflineConversion()
{
    IBPNGClient* client = getBPNGClient();

    // We use the sample Offline Data Set that was downloaded
    // with sampleFunctionDownload().
    // Its up to you to ensure an existing file.
    BOOL ret = client->initOffline("..\\testoutdir\\BP2_Offline.zip");
    if (ret == 0)
    {
        BPNGError err = client->getLastError();
        cout << "Failed to init offline." << endl;
        cout << "BPNGErrCode: " << err.code << ", " << err.msg << endl;
        client->release();
        return;
    }

    IRdbEventList* list = client->getEventList();
    RdbEventList eventList(list);
    if (eventList.size() == 0)
    {
        cout << "Empty event list" << endl;
        client->release();
        return;
    }

    uint64_t markerTimeStamp = 0;
    // search last marker
    for (int i = eventList.size() - 1; i >= 0; --i)
    {

```

```

        if (eventList[i].type == EVID_MARKER)
        {
            markerTimeStamp = eventList[i].timeStamp;
            break;
        }
    }

    if (markerTimeStamp == 0)
    {
        cout << "No marker found." << endl;
        client->release();
        return;
    }

    // Ensure the out directory exists
    ret = _mkdir("../testoutdir");
    if (ret != 0 && errno != EEXIST)
    {
        cout << "Failed to create output directory" << endl;
        client->release();
        return;
    }

    // Get a conversion set
    IConversionSet* conversionSet = client->createNewConversionSet();

    // The time span has to be 60s before and 60s after the marker
    uint64_t startTime = markerTimeStamp - 60 * 1000000; // in usec
    uint64_t endTime = markerTimeStamp + 60 * 1000000; // in usec

    // If you want to convert more than one span,
    // call this function several times
    conversionSet->addTimeSpan(startTime, endTime);

    // CAN #1 and CAN #2 are supposed to be written to one asc output file
    // each.
    // CAN #3 and CAN #4 are supposed to be written together in another asc
    // file.
    // All other CAN channels are supposed to be written together in one
    // BLF file.
    const IChannelList* channels = client->getLoggerChannels();
    for (int i = 0; i < channels->getSize(); ++i)
    {
        ChannelType type = channels->getChannel(i)->getType();
        if (type != CH_CANLS && type != CH_CANHS)
            continue;

        // Note: channel indices are zero based
        int index = channels->getChannel(i)->getIndex();
        if (index == 0 || index == 1)
        {
            // CAN #1 and #2 in separate files
            // -1 as fileId parameter creates a separate file for
            this channel
            conversionSet->addChannel(type, index, CANOE, -1);
        }
        else if (index == 2 || index == 3)
        {
            // CAN #3 and #4 in the same file.
            // fileId != -1 will write all channels with the same
            format and same
            // file Id to the same output file (if procurable in
            accordance with
            // the format specification.
            conversionSet->addChannel(type, index, CANOE, 10);
        }
        else

```

```

        {
            // All other CAN channels to one BLF file.
            conversionSet->addChannel(type, index, BLF, 20);
        }
    }

    ret = client->convertData(conversionSet, "..\\testoutdir");
    if (ret == 0)
    {
        BPNGError err = client->getLastError();
        cout << "Failed to convert data." << endl;
        cout << "BPNGErrCode: " << err.code << ", " << err.msg << endl;
        client->release();
        return;
    }

    // free memory
    client->release();
}

// This function shows how to:
// - download the configuration
// - reconfigure the logger device
// - set the default config
void sampleFunctionConfiguration(BP2Device device)
{
    IBPNGClient* client = getBPNGClient();

    BOOL ret = client->connectLogger(device.ipAddress.c_str());
    if (ret == 0)
    {
        BPNGError err = client->getLastError();
        cout << "Failed to connect logger." << endl;
        cout << "BPNGErrCode: " << err.code << ", " << err.msg << endl;
        client->release();
        return;
    }

    // save current config
    string targetPath = "..\\testoutdir\\Config_" + device.name + ".zip";
    ret = client->getConfig(targetPath.c_str());
    if (ret == 0)
    {
        BPNGError err = client->getLastError();
        cout << "Failed to download configuration." << endl;
        cout << "BPNGErrCode: " << err.code << ", " << err.msg << endl;
        client->disconnectLogger();
        client->release();
        return;
    }

    // here you could change the downloaded configuration
    // by extracting it and modifying the xml files
    // see documentation of IBPNGClient::getConfig()
    // or IBPNGClient::reconfigLogger().

    // We use the same config that we downloaded
    string newConfigPath = targetPath;
    ret = client->reconfigLogger(newConfigPath.c_str());
    if (ret == 0)
    {
        BPNGError err = client->getLastError();
        cout << "Failed to reconfigure the logger." << endl;
        cout << "BPNGErrCode: " << err.code << ", " << err.msg << endl;
        client->disconnectLogger();
        client->release();
        return;
    }
}

```

```
}

// Setting the default config
ret = client->setDefaultConfig();
if (ret == 0)
{
    BPNGError err = client->getLastError();
    cout << "Failed to set default config to the logger." << endl;
    cout << "BPNGErrorCode: " << err.code << ", " << err.msg << endl;
    client->disconnectLogger();
    client->release();
    return;
}

// disconnect
client->disconnectLogger();
// free memory
client->release();
}
```

## Kapitel 2

# Module Index

### 2.1 Modules

Here is a list of all modules:

Download Functions . . . . .	15
Conversion Functions . . . . .	16

## Kapitel 3

# Class Index

### 3.1 Class Hierarchy

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

BP2Device . . . . .	17
BPNGError . . . . .	17
CANPseudoMessagesProperties . . . . .	21
ClientProperties . . . . .	23
CommonProperties . . . . .	24
DataSpan . . . . .	25
IBPNGClient . . . . .	25
IBPNGClientListener . . . . .	37
BPNGLoggerDetector . . . . .	18
IChannel . . . . .	41
IChannelList . . . . .	41
IConversionSet . . . . .	42
IFormatInfo . . . . .	43
IFormatList . . . . .	44
IRdbEvent . . . . .	45
IRdbEventList . . . . .	45
MOSTPseudoMessagesProperties . . . . .	46
OnlineLoggerInfo . . . . .	46
RdbEvent . . . . .	47
RdbEventList . . . . .	48

## Kapitel 4

# Class Index

### 4.1 Class List

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

<a href="#">BP2Device</a>	17
<a href="#">BPNGError</a>	
Error struct with error code and optional error message	17
<a href="#">BPNGLoggerDetector</a>	18
<a href="#">CANPseudoMessagesProperties</a>	
CAN pseudo messages can be written to the CANoe ASC format	21
<a href="#">ClientProperties</a>	
The <a href="#">ClientProperties</a> are a combination of <a href="#">CommonProperties</a> , <a href="#">CanPseudo-</a> <a href="#">MessageProperties</a> and <a href="#">MostPseudoMessageProperties</a>	23
<a href="#">CommonProperties</a>	
Common properties	24
<a href="#">DataSpan</a>	25
<a href="#">IBPNGClient</a>	
Interface class for the blue PiraT 2 client library	25
<a href="#">IBPNGClientListener</a>	37
<a href="#">IChannel</a>	
Channel interface	41
<a href="#">IChannelList</a>	
Channel list interface	41
<a href="#">IConversionSet</a>	
A conversion set stores all conversion relevant settings	42
<a href="#">IFormatInfo</a>	
FormatInfo interface	43
<a href="#">IFormatList</a>	
Format list interface	44
<a href="#">IRdbEvent</a>	
Interface to an RDB event	45
<a href="#">IRdbEventList</a>	
Interface to a list of rdb events	45
<a href="#">MOSTPseudoMessagesProperties</a>	
MOST pseudo messages for each trigger can be written to MOST formats	46
<a href="#">OnlineLoggerInfo</a>	
Struct with information about a logger found in LAN	46

<a href="#">RdbEvent</a>	
Implementation class for a wrapper of <a href="#">IRdbEvent</a> using STL classes . . . . .	47
<a href="#">RdbEventList</a>	
Implementation class for a wrapper of <a href="#">IRdbEventList</a> using STL classes . . . . .	48



## Kapitel 5

# File Index

### 5.1 File List

Here is a list of all documented files with brief descriptions:

<a href="#">BPNGDefines.h</a>	
Defines for blue PiraT 2 Library . . . . .	50
<b>BPNGLoggerDetector.hh</b> . . . . .	??
<a href="#">IBPNGClient.h</a>	
Interface class for the BPNGClient DLL . . . . .	57
<a href="#">IBPNGClientListener.h</a>	
Interface class for the BPNGClient listener . . . . .	58
<b>RdbEventList.hh</b> . . . . .	??

## Kapitel 6

# Module Documentation

### 6.1 Download Functions

This are all function needed for downloading an offline data set.

#### Functions

- virtual BOOL WINAPI [IBPNGClient::connectLogger](#) (const char \*ipAddress)=0  
*Connect to logger with passed IP address.*

#### 6.1.1 Detailed Description

This are all function needed for downloading an offline data set.

#### 6.1.2 Function Documentation

**6.1.2.1** virtual BOOL WINAPI IBPNGClient::connectLogger ( const char \* *ipAddress* ) [pure virtual]

Connect to logger with passed IP address.

,

```
While the logger is connected, it won't go to standby mode until the last
IBPNGClient instance is disconnected.
If connect fails the function will return 0. On success the return value is 1.
In case of failure further information can be retrieved with GetLastError().
```

```
@param ipAddress IP address of the logger that should be connected
@return 0 on failure, 1 on success
```

## 6.2 Conversion Functions

This are all function needed for converting data.

This are all function needed for converting data.

## Kapitel 7

# Class Documentation

### 7.1 BP2Device Struct Reference

#### Public Member Functions

- **BP2Device** ([OnlineLoggerInfo](#) \*logger)

#### Public Attributes

- std::string **ipAddress**
- std::string **name**
- std::string **mainboardNr**
- bool **connected**
- std::string **connectedUser**
- int **status**

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

- BPNGLoggerDetector.hh

### 7.2 BPNGError Struct Reference

Error struct with error code and optional error message.

```
#include <BPNGDefines.h>
```

#### Public Attributes

- [BPNGErrCode](#) *code*
- const char \* [msg](#)  
*error message*

### 7.2.1 Detailed Description

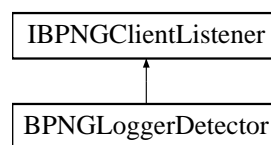
Error struct with error code and optional error message.

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

- [BPNGDefines.h](#)

## 7.3 BPNGLoggerDetector Class Reference

Inheritance diagram for BPNGLoggerDetector:



### Public Member Functions

- `std::vector< BP2Device > getLoggerList (unsigned searchTimeOut)`
- `virtual void WINAPI onBPNGDeviceDetected (OnlineLoggerInfo *info)`  
*Called to notify a detected logger in network.*
- `virtual void WINAPI onBPNGDeviceDisappeared (OnlineLoggerInfo *info)`  
*Called to notify a disappeared logger.*
- `virtual void WINAPI onBPNGDeviceStateChange (OnlineLoggerInfo *info)`  
*Called to notify a logger's state change.*
- `virtual int WINAPI onProgressDataDownload (int percentCompleted)`  
*Called to indicate the current progress of a file transfer.*
- `virtual int WINAPI onProgressConversion (int percentCompleted, const char *status)`  
*Called to indicate the current progress of file conversion.*
- `virtual void WINAPI onStatusMessage (const char *statusMsg)`  
*Called to send additional information of the current process to the calling app.*
- `virtual void WINAPI onWarning (BPNGWarningCode warningCode, const char *warnMsg)`  
*Called to inform about a warning.*
- `virtual int WINAPI onTargetPathTooLong (char *newTarget, int maxSize)`  
*Called on a too long target directory.*
- `virtual int WINAPI getOverwritingPermission (const char *filePath)`  
*Called on existing output trace files.*
- `virtual int WINAPI onCriticalDiskSpace (uint64_t freeSpace, uint64_t neededSpace, const char *drive, const char *msg)`  
*Called in case of not enough free disk space.*
- `virtual void WINAPI onFirmwareUpdateProgress (int percentage, int stepId, int subStepId, const char *desc)`  
*Called on firmware update progress.*
- `virtual void WINAPI onFirmwareUpdateError (int errorId)`
- `virtual int WINAPI onGetLogReportProgress (int percentage, const char *desc)`

### 7.3.1 Member Function Documentation

**7.3.1.1** virtual int WINAPI BPNGLoggerDetector::getOverwritingPermission ( const char \* *filePath* ) [inline, virtual]

Called on existing output trace files.

When an output trace file already exists this function is called. The listener has the possibility to return one of following values: -1: no, don't overwrite file -2: no, overwrite neither this nor any following file 1: yes, overwrite file 2: yes, overwrite this and all following files 0: cancel conversion

Implements [IBPNGClientListener](#).

**7.3.1.2** virtual void WINAPI BPNGLoggerDetector::onBPNGDeviceDetected ( [OnlineLoggerInfo](#) \* *info* ) [virtual]

Called to notify a detected logger in network.

All char\* of the passed [OnlineLoggerInfo](#)\* are only valid for the time of the function call. Please ensure to copy the string values.

Implements [IBPNGClientListener](#).

**7.3.1.3** virtual void WINAPI BPNGLoggerDetector::onBPNGDeviceDisappeared ( [OnlineLoggerInfo](#) \* *info* ) [virtual]

Called to notify a disappeared logger.

All char\* of the passed [OnlineLoggerInfo](#)\* are only valid for the time of the function call. Please ensure to copy the string values.

Implements [IBPNGClientListener](#).

**7.3.1.4** virtual void WINAPI BPNGLoggerDetector::onBPNGDeviceStateChange ( [OnlineLoggerInfo](#) \* *info* ) [virtual]

Called to notify a logger's state change.

All char\* of the passed [OnlineLoggerInfo](#)\* are only valid for the time of the function call. Please ensure to copy the string values.

Implements [IBPNGClientListener](#).

**7.3.1.5** virtual int WINAPI BPNGLoggerDetector::onCriticalDiskSpace ( uint64\_t *freeSpace*, uint64\_t *neededSpace*, const char \* *drive*, const char \* *msg* ) [inline, virtual]

Called in case of not enough free disk space.

This notifies the listener about not enough free disk space for data download or conversion. The user can continue or abort the process. Returning 0 will abort the process. In some cases continuing without providing more disk space will call this function immediately again.

## Parameters

<i>freeSpace</i>	Amount of free space
<i>neededSpace</i>	Amount of needed space
<i>drive</i>	Name of the drive where to store data
<i>msg</i>	Additional message to display

## Returns

return 0 when process should be aborted, 1 to ignore

Implements [IBPNGClientListener](#).

**7.3.1.6** virtual int WINAPI BPNGLoggerDetector::onGetLogReportProgress ( int *percentage*, const char \* *desc* ) [inline, virtual]

Called on creation of log report

## Returns

return value 0 indicates an abort request from the implementing class

Implements [IBPNGClientListener](#).

**7.3.1.7** virtual int WINAPI BPNGLoggerDetector::onProgressConversion ( int *percentCompleted*, const char \* *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.

## Parameters

<i>percent-Completed</i>	Percent of the entire conversion process (from 0...100%), -1 indicates the same value as from last function call
<i>status</i>	Status of the conversion process (e.g. "Converting trace data. Block 5 of 32")

## Returns

return value 0 indicates an abort request from the implementing class

Implements [IBPNGClientListener](#).

**7.3.1.8** virtual int WINAPI BPNGLoggerDetector::onProgressDataDownload ( int *percentCompleted* ) [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.

## Parameters

<i>percent-Completed</i>	Percentage of the entire download process (from 0...100%)
--------------------------	---

## Returns

return value 0 indicates an abort request from the implementing class

Implements [IBPNGClientListener](#).

**7.3.1.9** `virtual void WINAPI BPNGLoggerDetector::onStatusMessage ( const char * statusMsg )`  
`[inline, virtual]`

Called to send additional information of the current process to the calling app.

This function transmit message strings to the listener class. Those messages are only for information purpose. The receiver doesn't have to react on it but can display it on the screen.

Implements [IBPNGClientListener](#).

**7.3.1.10** `virtual int WINAPI BPNGLoggerDetector::onTargetPathTooLong ( char * newTarget, int maxSize )`  
`[inline, virtual]`

Called on a too long target directory.

Called when the resulting file name of the converted files is longer than the maximum allowed size of the file system (Windows 260). The lib user has to pass a new (shorter) base target directory to the passed char array with strcpy. The memory of the array is already allocated by the library and it's size is maxSize. When a new directory was set the value 1 must be returned. If a listener wants to ignore this function call 0 has to be returned.

Implements [IBPNGClientListener](#).

**7.3.1.11** `virtual void WINAPI BPNGLoggerDetector::onWarning ( BPNGWarningCode warningCode, const char * warnMsg )`  
`[inline, virtual]`

Called to inform about a warning.

This function transmit a warning message to the listener class. Warnings have a WARNING\_CODE and a warning message. Warnings do not interrupt the current process but should be noticed from the user to possibly initiate further provisions.

Implements [IBPNGClientListener](#).

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

- BPNGLoggerDetector.hh

## 7.4 CANPseudoMessagesProperties Struct Reference

CAN pseudo messages can be written to the CANoe ASC format.

```
#include <BPNGDefines.h>
```



## Public Attributes

- **BOOL** [canTimeStampMessage](#)  
*Active flag for writing periodical CAN pseudo messages with absolute time stamps.*
- **int** [channelTS](#)  
*CAN channel for the time stamp pseudo messages.*
- **int** [dlcTS](#)  
*DLC for the time stamp pseudo messages.*
- **int** [canIdTS](#)  
*CAN ID for the time stamp pseudo messages.*
- **int** [hourBitPos](#)  
*Bit position for the hour (0..23, 5 bit length) value in the CAN data bytes.*
- **int** [minuteBitPos](#)  
*Bit position for the minute (0..59, 6 bit length) value in the CAN data bytes.*
- **int** [secondBitPos](#)  
*Bit position for the second (0..59, 6 bit length) value in the CAN data bytes.*
- **int** [dayBitPos](#)  
*Bit position for the day (0..31, 5 bit length) value in the CAN data bytes.*
- **int** [monthBitPos](#)  
*Bit position for the month (0..12, 4 bit length) value in the CAN data bytes.*
- **int** [yearBitPos](#)  
*Bit position for the year (8 bit length) value in the CAN data bytes.*
- **BOOL** [canTriggerMessage](#)  
*Active flag for writing CAN pseudo messages with trigger information.*
- **int** [channelTrigger](#)  
*CAN channel for the trigger pseudo messages.*
- **int** [dlcTrigger](#)  
*DLC for the trigger pseudo messages.*
- **int** [canIdTrigger](#)  
*CAN ID for the trigger pseudo messages.*
- **int** [triggerNumBitPos](#)  
*Bit position for the trigger's index (16 bit length)*
- **int** [reserved1](#)  
*reserved*
- **int** [reserved2](#)  
*reserved*
- **int** [reserved3](#)  
*reserved*
- **int** [reserved4](#)  
*reserved*

### 7.4.1 Detailed Description

CAN pseudo messages can be written to the CANoe ASC format.

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

- [BPNGDefines.h](#)

## 7.5 ClientProperties Struct Reference

The [ClientProperties](#) are a combination of [CommonProperties](#), [CanPseudoMessageProperties](#) and [MostPseudoMessageProperties](#).

```
#include <BPNGDefines.h>
```

### Public Attributes

- [CommonProperties common](#)  
*The properties object for all general properties,.*
- [CANPseudoMessagesProperties canPseudoMessages](#)  
*The properties object for the CAN pseudo messages,.*
- [MOSTPseudoMessagesProperties mostPseudoMessages](#)  
*The properties object for the MOST pseudo messages,.*

### 7.5.1 Detailed Description

The [ClientProperties](#) are a combination of [CommonProperties](#), [CanPseudoMessageProperties](#) and [MostPseudoMessageProperties](#).

### 7.5.2 Member Data Documentation

#### 7.5.2.1 CANPseudoMessagesProperties ClientProperties::canPseudoMessages

The properties object for the CAN pseudo messages,.

See also

[CANPseudoMessagesProperties](#)

#### 7.5.2.2 CommonProperties ClientProperties::common

The properties object for all general properties,.

See also

[CommonProperties](#)

#### 7.5.2.3 MOSTPseudoMessagesProperties ClientProperties::mostPseudoMessages

The properties object for the MOST pseudo messages,.

See also

[MOSTPseudoMessagesProperties](#)

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

- [BPNGDefines.h](#)

## 7.6 CommonProperties Struct Reference

Common properties.

```
#include <BPNGDefines.h>
```

### Public Attributes

- char \* [nameOfTester](#)  
*Name of tester that is written to the converted file names.*
- int [maxOutputSizeMB](#)  
*Maximum file size for converted files. When this size is reached a new file is created.*
- BOOL [separatedTimeFormat](#)
- BOOL [separatedTimeFormatInOfflineSet](#)
- char \* [alternativeLoggerName](#)  
*The logger device's name is included in the converted files' names. An alternative logger name can be used.*
- BOOL [useAlternativeLoggerName](#)  
*Set this field to 1 if the alternative logger name should be used in converted file names, 0 if not.*
- BOOL [useSubdirectories](#)  
*Set to 1 if converted files should be stored in subdirectories named by their start date, set 0 if they should not.*
- BOOL [midnightSplitting](#)  
*Set to 1 if converted files should be splitted at 00:00:00 of each date, set to 0 if they should not.*
- BOOL [fileTimeSpansLikeSelection](#)
- BOOL [markerNumbersInFileNames](#)  
*Specifies whether the indices of the marker included in a converted file should be appended to its file name.*
- BOOL [subfolderWithLoggerName](#)
- int [reserved1](#)  
*reserved*
- int [reserved2](#)  
*reserved*
- int [reserved3](#)  
*reserved*
- int [reserved4](#)  
*reserved*

### 7.6.1 Detailed Description

Common properties.

### 7.6.2 Member Data Documentation

#### 7.6.2.1 BOOL CommonProperties::fileTimeSpansLikeSelection

The file names of the converted files contain the time span of the included data. Setting this parameter to 1 will create time spans like they were specified in the [IConversionSet](#). Setting this to 0 will create time spans according to the effectively included data.

### 7.6.2.2 BOOL CommonProperties::separatedTimeFormat

Specifies the time format that should be used for converted files. Set 1 for long format (e.g. [2011-12-20]\_10.15.48) or 0 for short format (e.g. 20111220\_101548)

### 7.6.2.3 BOOL CommonProperties::separatedTimeFormatInOfflineSet

Specifies the time format that should be used for offline conversion sets. Set 1 for long format (e.g. [2011-12-20]\_10.15.48) or 0 for short format (e.g. 20111220\_101548)

### 7.6.2.4 BOOL CommonProperties::subfolderWithLoggerName

Specifies whether the name of the subfolder the converted files are stored in should contain the logger name or not.

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

- [BPNGDefines.h](#)

## 7.7 DataSpan Struct Reference

### Public Attributes

- [uint8\\_t type](#)  
*set type to 0 for a id based range, set type to 1 for a time based range*
- [uint64\\_t start](#)  
*start time/id of data span*
- [uint64\\_t end](#)  
*end time/id of data span*

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

- [BPNGDefines.h](#)

## 7.8 IBPNGClient Struct Reference

Interface class for the blue PiraT 2 client library.

```
#include <IBPNGClient.h>
```

### Public Member Functions

- virtual void WINAPI [scanNetworkForLogger](#) ()=0  
*Scan network for logger.*
- virtual BOOL WINAPI [connectLogger](#) (const char \*ipAddress)=0  
*Connect to logger with passed IP address.*

- virtual void WINAPI [disconnectLogger](#) ()=0  
*Disconnect the currently connected logger.*
- virtual BOOL WINAPI [isConnected](#) ()=0  
*Check the logger connection, returns 1 for valid connection and 0 for no or broken connection.*
- virtual BOOL WINAPI [initOnline](#) ()=0  
*Initialisation of download and online conversion process.*
- virtual BOOL WINAPI [initOffline](#) (const char \*path)=0  
*Initialisation of offline conversion process.*
- virtual int WINAPI [downloadDataSpans](#) (uint16\_t numSpans, [DataSpan](#) \*dataSpans, const char \*target, BOOL doSorting)=0  
*Download trace data.*
- virtual [IConversionSet](#) \*WINAPI [createNewConversionSet](#) ()=0  
*Returns the pointer to a new conversion set.*
- virtual int WINAPI [convertData](#) ([IConversionSet](#) \*conversionSet, const char \*target)=0  
*Convert all data specified by conversionSet.*
- virtual BOOL WINAPI [getConfig](#) (const char \*path)=0  
*Download the current logger configuration to the passed path.*
- virtual BOOL WINAPI [reconfigLogger](#) (const char \*configZip)=0  
*Reconfig logger with the zipped new configuration.*
- virtual BOOL WINAPI [setDefaultConfig](#) ()=0  
*Reconfig logger with the default configuration.*
- virtual [IRdbEventList](#) \*WINAPI [getEventList](#) ()=0  
*Get list of all events from the RDB.*
- virtual const char \*WINAPI [getInstanceName](#) ()=0  
*Return the instance name passed to the [getBPNGClient\(\)](#) function.*
- virtual const char \*WINAPI [getReferenceDataBasePath](#) ()=0  
*Get path to the reference data base.*
- virtual const char \*WINAPI [getConfigPath](#) ()=0  
*Get path to the config directory (after calling one of the init functions)*
- virtual const char \*WINAPI [getDeviceName](#) ()=0  
*Get name of device.*
- virtual const [IChannelList](#) \*WINAPI [getLoggerChannels](#) ()=0  
*Returns pointer to a channel list interface.*
- virtual const char \*WINAPI [getVersions](#) ()=0  
*Get the firmware and hardware version string.*
- virtual BOOL WINAPI [updateFirmware](#) (const char \*fwPath, BOOL force)=0  
*Update firmware.*
- virtual BOOL WINAPI [updateLicenses](#) (const char \*licenseFilePath)=0  
*Update licenses.*
- virtual const char \*WINAPI [getLicenses](#) ()=0  
*Returns the license file's content as string.*
- virtual BOOL WINAPI [removeAllLicenses](#) ()=0  
*Removes the current license file from the logger.*
- virtual int WINAPI [deleteData](#) (uint16\_t numSpans, [DataSpan](#) \*dataSpans)=0  
*Delete trace data.*
- virtual int WINAPI [deleteAllData](#) ()=0  
*Delete all trace data on the logger.*

- virtual BOOL WINAPI [setInfoEvent](#) (const char \*msg)=0  
*Set an info event with the passed string on the connected logger.*
- virtual BOOL WINAPI [setMarker](#) ()=0  
*Set a marker on the connected logger. Returns 0 on error.*
- virtual uint64\_t WINAPI [getCurrentLoggerTime](#) ()=0  
*Returns the current loggertime in seconds since 01.01.1970 UTC.*
- virtual BOOL WINAPI [setTime](#) (uint64\_t time)=0  
*Set logger time and date to the passed UTC time stamp.*
- virtual void WINAPI [keepLoggerAlive](#) (const char \*ip)=0  
*Call this to keep logger alive.*
- virtual void WINAPI [stopKeepLoggerAlive](#) (const char \*ip)=0  
*Called to stop sending keep alive pings to the logger specified via the passed ip.*
- virtual [IFormatList](#) \*WINAPI [getAvailableFormats](#) ()=0  
*Return pointer to a format list interface. Returns null in case of error.*
- virtual void WINAPI [flashDeviceLED](#) ()=0  
*Let the connected device blink its front LEDs for identification.*
- virtual int WINAPI [createCCPXCFiles](#) (const char \*dbcDir, const char \*xsdDir, const char \*xmlDir)=0  
*and a Vector DBC file for each device*
- virtual void WINAPI [addListener](#) ([IBPNGClientListener](#) \*listener)=0  
*Add a listener.*
- virtual void WINAPI [removeListener](#) ([IBPNGClientListener](#) \*listener)=0  
*Remove a listener.*
- virtual [BPNGError](#) WINAPI [getLastError](#) ()=0  
*Get last error code.*
- virtual int WINAPI [getNumConversionErrors](#) ()=0  
*Returns the number of errors occurred during the last conversion process.*
- virtual [BPNGError](#) WINAPI [getConversionError](#) (int index)=0  
*Returns the conversion error at index.*
- virtual int WINAPI [getNumDownloadErrors](#) ()=0  
*Returns the number of errors occurred during the last download process.*
- virtual [BPNGError](#) WINAPI [getDownloadError](#) (int index)=0  
*Returns the download error at index.*
- virtual const char \*WINAPI [getLibVersion](#) ()=0  
*Returns the current client library version.*
- virtual void WINAPI [release](#) ()=0  
*Free memory of this [IBPNGClient](#) instance.*
- virtual [ClientProperties](#) \*WINAPI [getProperties](#) ()=0  
*Returns a pointer to the current properties.*
- virtual void WINAPI [setProperties](#) ([ClientProperties](#) \*val)=0  
*Set client properties.*
- virtual void WINAPI [assignDBCFile](#) (int channelIndexCAN, const char \*dbcFilePath)=0  
*Assign a DBC file to a CAN channel.*
- virtual int WINAPI [resetMarkerCounter](#) ()=0  
*Reset marker counter.*
- virtual int WINAPI [downloadBugReport](#) (const char \*targetPath, [BPNGBugreportMode](#) mode, uint64\_t startTime, uint64\_t endTime)=0  
*Download bug report.*

### 7.8.1 Detailed Description

Interface class for the blue PiraT 2 client library.

[IBPNGClient](#) is the interface class of the blue PiraT Client library. To get access to a blue PiraT 2 data logger you need a pointer to an implementing instance of the [IBPNGClient](#) interface. Use [getBPNGClient\(\)](#) to get such a pointer. This will create an implementing instance on the heap. To avoid conflict between different runtime libraries it is obligatory to release this object with its [IBPNGClient::release\(\)](#) function when not needed any more. Don't call the delete operator directly on this pointer.

### 7.8.2 Member Function Documentation

**7.8.2.1** `virtual void WINAPI IBPNGClient::assignDBCFile ( int channelIndexCAN, const char * dbcFilePath )` [pure virtual]

Assign a DBC file to a CAN channel.

#### Parameters

<i>channelIndexCAN</i>	Zero based CAN channel index
<i>dbcFilePath</i>	Absolute path to the dbc file

**7.8.2.2** `virtual int WINAPI IBPNGClient::convertData ( IConversionSet * conversionSet, const char * target )` [pure virtual]

Convert all data specified by conversionSet.

Before data from a logger or an offline data set can be converted, [IBPNGClient::initOnline\(\)](#) resp. [IBPNGClient::initOffline\(\)](#) must have been called before.

The data specified by conversionSet is converted to the passed target directory.

Function will return 0 on failure, 1 on success and -1 on user abort. In case of failure further information can be retrieved with [getLastError\(\)](#).

If [getLastError\(\)](#) returns BPNG\_CONVERSION\_ERRORS several errors occurred. Use [getNumConversionErrors\(\)](#) and [getConversionError\(int index\)](#) for detailed information.

#### Parameters

<i>conversionSet</i>	conversion settings, see <a href="#">IConversionSet</a>
<i>target</i>	target directory for the converted trace files. Depending on the passed <a href="#">ClientProperties</a> the files may be stored in sub folders named by date

**Returns**

0 on failure, 1 on success and -1 on user abort.

**7.8.2.3** `virtual int WINAPI IBPNGClient::createCCPXCPFiles ( const char * dbcDir, const char * xsdDir, const char * xmlDir )` [pure virtual]

and a Vector DBC file for each device

Parse CCPXCPMeasurement.xml and CCPXCPConfiguration.xml and create CCPXCPSequence.xml

**7.8.2.4** `virtual int WINAPI IBPNGClient::deleteAllData ( )` [pure virtual]

Delete all trace data on the logger.

In case of failure further information can be retrieved with [getLastError\(\)](#).

**Returns**

0 on failure, 1 on success and -1 on user abort.

**7.8.2.5** `virtual int WINAPI IBPNGClient::deleteData ( uint16_t numSpans, DataSpan * dataSpans )` [pure virtual]

Delete trace data.

Pass the size and the pointer to an array of [DataSpan](#). Each span specifies either a time span or an index span from the reference data base's entry IDs (DataBaseEntryId). If you want to create spans with those IDs you have to call the [initOnline\(\)](#) function first to get the current RDB file.

Function will return 0 on failure, 1 on success and -1 on user abort. In case of failure further information can be retrieved with [getLastError\(\)](#).

**Parameters**

<i>numSpans</i>	Size of the passed <a href="#">DataSpan</a> array in second parameter
<i>dataSpans</i>	Array of <a href="#">DataSpan</a> , specifying the time or ID spans that should be downloaded

**Returns**

0 on failure, 1 on success and -1 on user abort.

**7.8.2.6** `virtual int WINAPI IBPNGClient::downloadBugReport ( const char * targetPath, BPNGBugreportMode mode, uint64_t startTime, uint64_t endTime )` [pure virtual]

Download bug report.

The downloaded bug report is a ZIP archive with several log data and system files for error analyzing purposes.



## Parameters

<i>targetPath</i>	Path inclusive file name under that the bug report will be stored.
<i>mode</i>	that specifies what kind of data should be included in the report,

## See also

[BPNGBugreportMode](#)

## Parameters

<i>startTime</i>	Start time for the time span of trace data that should be included (usec since 01.01.1970 UTC). Only for mode BR_FULL_ALL_TRACES and BR_FULL_TIMESPAN_TRACES
<i>endTime</i>	End time for the time span of trace data that should be included (usec since 01.01.1970 UTC). Only for mode BR_FULL_ALL_TRACES and BR_FULL_TIMESPAN_TRACES

## Returns

0 on failure, 1 on success and -1 on user abort.

**7.8.2.7** `virtual int WINAPI IBPNGClient::downloadDataSpans ( uint16_t numSpans, DataSpan * dataSpans, const char * target, BOOL doSorting )` [pure virtual]

Download trace data.

Pass the size and the pointer to an array of [DataSpan](#). Each span specifies either a time span or an index span from the reference data base's entry IDs (DataBaseEntryId). [IBPNGClient::init-Online\(\)](#) must have been called before.

Function will return 0 on failure, 1 on success and -1 on user abort. In case of failure further information can be retrieved with [getLastError\(\)](#).

If [getLastError\(\)](#) returns BPNG\_DOWNLOAD\_ERRORS several errors occurred. Use [getNumDownloadErrors\(\)](#) and [getDownloadError\(int index\)](#) for detailed information.

## Parameters

<i>numSpans</i>	Size of the passed <a href="#">DataSpan</a> array in second parameter
<i>dataSpans</i>	Array of <a href="#">DataSpan</a> , specifying the time or ID spans that should be downloaded
<i>target</i>	Path to the target directory or ZIP file. A passed directory must be empty or not existing. A passed ZIP path must not exist.
<i>doSorting</i>	Specifies whether the traces from different logger-internal sources should be sorted to one output stream or not.

## Returns

0 on failure, 1 on success and -1 on user abort.

**7.8.2.8** virtual void WINAPI IBPNGClient::flashDeviceLED ( ) [pure virtual]

Let the connected device blink its front LEDs for identification.

You can use this function to identify you device if you can't identify it over the Name or IP address given from the [IBPNGClientListener::onBPNGDeviceDetected](#) callback function.

**7.8.2.9** virtual IFormatList\* WINAPI IBPNGClient::getAvailableFormats ( ) [pure virtual]

Return pointer to a format list interface. Returns null in case of error.

All formats returned by this function are available for data conversion.

See also

[IFormatList](#), [IFormatInfo](#)

**7.8.2.10** virtual BOOL WINAPI IBPNGClient::getConfig ( const char \* *path* ) [pure virtual]

Download the current logger configuration to the passed path.

If you download the current configuration from the data logger you get a zip Archive that contains all relevant XML and XSD files to modify the configuration in a valid way and reconfigure the device with the [reconfigLogger\(\)](#) function.

Please note: It is up to you to ensure a valid configuration if you want to modify it with your own tools. You should only modify the xml and not the xsd files. "DeviceConfiguration.xml" and "FirmwareConfiguration.xml" should also not be modified. They specify all xml files that are mandatory to reconfigure the data logger. You can validate the xml files with the supplied xsd files and a XML library of your choice. One possibility would be the XERCES library, see <http://xerces.apache.org/xerces-c/>

#### Parameters

<i>path</i>	The path inclusive file name where to store the downloaded configuration ZIP file.
-------------	--

**7.8.2.11** virtual BPNGError WINAPI IBPNGClient::getConversionError ( int *index* ) [pure virtual]

Returns the conversion error at *index*.

After getting the number of conversion errors with [getNumConversionErrors\(\)](#) you can get all single errors with this function.

**7.8.2.12** virtual const char\* WINAPI IBPNGClient::getDeviceName ( ) [pure virtual]

Get name of device.

After calling one of the init functions [IBPNGClient::initOnline\(\)](#) or [IBPNGClient::initOffline\(\)](#) this function returns the currently configured device name.

**Returns**

The device name

**See also**

[initOnline\(\)](#), [initOffline\(\)](#)

**7.8.2.13** virtual **BPNGError** WINAPI **IBPNGClient::getDownloadError** ( int *index* ) [pure virtual]

Returns the download error at *index*.

After getting the number of download errors with [getNumDownloadErrors\(\)](#) you can get all single errors with this function.

**7.8.2.14** virtual **IRdbEventList\*** WINAPI **IBPNGClient::getEventList** ( ) [pure virtual]

Get list of all events from the RDB.

If [initOnline\(\)](#) was called before, the events of the logger's RDB is returned. If [initOffline\(\)](#) was called before, the events of the RDB included in the offline data set is returned.

**Returns**

Pointer to a [IRdbEventList](#)

**7.8.2.15** virtual **BPNGError** WINAPI **IBPNGClient::getLastError** ( ) [pure virtual]

Get last error code.

If any called BPNGClient function returns a value that indicates an error you can retrieve further information about that error with this function.

**Returns**

The error description with error code and optional string value.

**See also**

[BPNGError](#)

**7.8.2.16** virtual const **IChannelList\*** WINAPI **IBPNGClient::getLoggerChannels** ( ) [pure virtual]

Returns pointer to a channel list interface.

After calling one of the init functions [IBPNGClient::initOnline\(\)](#) or [IBPNGClient::initOffline\(\)](#) this function returns a pointer to the logger's resp. offline data set's channel list.

In case of error null is returned and further information can be retrieved with [getLastError\(\)](#).

See also

[IChannelList](#)

**7.8.2.17** `virtual int WINAPI IBPNGClient::getNumConversionErrors ( ) [pure virtual]`

Returns the number of errors occurred during the last conversion process.

If [convertData\(\)](#) fails, [getLastError\(\)](#) can return different kinds of errors. There are types of errors that won't interrupt the conversion process but will be gathered during conversion and notified at the end. In that case the error code returned by [getLastError\(\)](#) will be `BPNG_CONVERSION_ERRORS` and you can get the number of errors with this function.

See also

[getConversionError\(\)](#)

**7.8.2.18** `virtual int WINAPI IBPNGClient::getNumDownloadErrors ( ) [pure virtual]`

Returns the number of errors occurred during the last download process.

If [downloadDataSpans\(\)](#) fails, [getLastError\(\)](#) can return different kinds of errors. There are types of errors that won't interrupt the download process but will be gathered during download and notified at the end. In that case the error code returned by [getLastError\(\)](#) will be `BPNG_DOWNLOAD_ERRORS` and you can get the number of errors with this function.

See also

[getDownloadError\(\)](#)

**7.8.2.19** `virtual ClientProperties* WINAPI IBPNGClient::getProperties ( ) [pure virtual]`

Returns a pointer to the current properties.

See also

[ClientProperties](#), [setProperties\(\)](#)

**7.8.2.20** `virtual const char* WINAPI IBPNGClient::getReferenceDataBasePath ( ) [pure virtual]`

Get path to the reference data base.

After calling one of the init functions [IBPNGClient::initOnline\(\)](#) or [IBPNGClient::initOffline\(\)](#) this function returns the path to the current Reference Data Base of the logger resp. the offline data set. For online processes, the RDB is downloaded from the logger to a tmp directory. For offline processes from a ZIP archive, the RDB is extracted to a tmp directory. For offline processes from a directory this function just returns the path to the RDB inside this directory.

**Returns**

Path to the downloaded or extracted RDB file

**See also**

[initOnline\(\)](#), [initOffline\(\)](#)

**7.8.2.21 virtual BOOL WINAPI IBPNGClient::initOffline ( const char \* *path* ) [pure virtual]**

Initialisation of offline conversion process.

For trace conversion from an offline data set this function must be called first.

Within this function the reference data base is read. Please note that reading a large RDB may take some time, espacially in debug mode.

Function will return 0 on failure and 1 on success. In case of failure further information can be retrieved with [getLastError\(\)](#).

**Returns**

0 on failure, 1 on success

**7.8.2.22 virtual BOOL WINAPI IBPNGClient::initOnline ( ) [pure virtual]**

Initialisation of download and online conversion process.

For trace download and conversion directly from the device this function must be called after the logger is connected.

Within this function the reference data base is downloaded and read. Please note that reading a large RDB may take some time, espacially in debug mode.

Function will return 0 on failure and 1 on success. In case of failure further information can be retrieved with [getLastError\(\)](#).

**Returns**

0 on failure, 1 on success

**7.8.2.23 virtual void WINAPI IBPNGClient::keepLoggerAlive ( const char \* *ip* ) [pure virtual]**

Call this to keep logger alive.

The blue PiraT 2 data logger can be configured to go to standby after a specified timeout without any bus traffic on the connected interfaces. If you want to have access to a device without bus traffic, and you don't want to connect to it with [connectLogger\(\)](#) you have to keep it alive by calling this function. This will start a separate thread that sends periodically ping messages to the passed IP address. Receiving these ping messages, the firmware will not shutdown the system.

**Parameters**

<i>ip</i>	The IP address of the logger that should be kept alive
-----------	--

See also

[stopKeepLoggerAlive\(\)](#)

**7.8.2.24** virtual BOOL WINAPI IBPNGClient::reconfigLogger ( const char \* *configZip* ) [pure virtual]

Reconfig logger with the zipped new configuration.

Reconfigures the logger with the passed configuration. The ZIP archive can be either one that was downloaded with the [getConfig\(\)](#) method, stored by the client software or a modified one. If you want to create your own configuration ZIP archive the structure of this file must be the same as of those mentioned above (xml files inside an "etc" directory). The abstract.txt file and all \*.xsd files are optional.

Please note: It is up to you to ensure a valid configuration if you want to modify it with your own tools. You should only modify the xml and not the xsd files. "DeviceConfiguration.xml" and "FirmwareConfiguration.xml" should also not be modified. They specify all xml files that are mandatory to reconfigure the data logger. You can validate the xml files with the supplied xsd files and a XML library of your choice. One possibility would be the XERCES library, see <http://xerces.apache.org/xerces-c/>

#### Parameters

<i>configZip</i>	Path to the zip file that contains the configuration.
------------------	---

#### Returns

0 on failure, 1 on success

**7.8.2.25** virtual void WINAPI IBPNGClient::release ( ) [pure virtual]

Free memory of this [IBPNGClient](#) instance.

With the call of [getBPNGClient\(\)](#) a new instance is created on the heap. The user is responsible to free its memory if it isn't needed any more. This function calls the delete operator on itself.

Important note: Any further function call on the [IBPNGClient](#) instance after [release\(\)](#) was called will cause a memory access violation and will crash the application!

**7.8.2.26** virtual void WINAPI IBPNGClient::scanNetworkForLogger ( ) [pure virtual]

Scan network for logger.

This function sends one broadcast UDP messages via all network adapters and notifies the calling application about responding devices with the listener functions onBPNGDeviceDetected(), onBPNGDeviceDisappeared() and onBPNGDeviceStateChange() (see [IBPNGClientListener.h](#)). For each broadcast message sent, the function waits for 100ms for responding devices

The first function call notifies about all found devices. All following calls on the same [IBPNGClient](#) instance will only notify about changes to the previous call.

**7.8.2.27 virtual BOOL WINAPI IBPNGClient::setDefaultConfig ( ) [pure virtual]**

Reconfig logger with the default configuration.

An invalid configuration will set the logger in error state. To fix this one possibility is to set the logger's default configuration.

**Returns**

0 on failure, 1 on success

**7.8.2.28 virtual BOOL WINAPI IBPNGClient::setInfoEvent ( const char \* msg ) [pure virtual]**

Set an info event with the passed string on the connected logger.

You can set an info event to the RDB. This event will be from type INFO and the passed message is written to the event's comment column

**Returns**

Returns 0 on failure, 1 on success

**7.8.2.29 virtual BOOL WINAPI IBPNGClient::setMarker ( ) [pure virtual]**

Set a marker on the connected logger. Returns 0 on error.

You can set an marker to the RDB. The set event will be from type MARKER.

**Returns**

Returns 0 on failure, 1 on success

**7.8.2.30 virtual void WINAPI IBPNGClient::setProperties ( ClientProperties \* val ) [pure virtual]**

Set client properties.

**See also**

[ClientProperties](#), [getProperties\(\)](#)

**7.8.2.31 virtual BOOL WINAPI IBPNGClient::setTime ( uint64\_t time ) [pure virtual]**

Set logger time and date to the passed UTC time stamp.

The parameter time must be in seconds since 01.01.1970 UTC

**Returns**

0 on failure, 1 on success

**7.8.2.32** virtual **BOOL** WINAPI **IBPNGClient::updateFirmware** ( const char \* *fwPath*, **BOOL** *force* )  
[pure virtual]

Update firmware.

This function updates the logger's firmware. An internal version check is done. If the second parameter *force* is 0 only firmware components with an older version than the component's version inside the firmware packet will be updated.

#### Parameters

<i>fwPath</i>	Path to the firmware packet file that should be installed.
<i>force</i>	Flag whether to update the components independently from the components' versions

#### Returns

0 on failure, 1 on success

**7.8.2.33** virtual **BOOL** WINAPI **IBPNGClient::updateLicenses** ( const char \* *licenseFilePath* )  
[pure virtual]

Update licenses.

Overwrites the current license file with the new one.

#### Parameters

<i>licenseFile-Path</i>	Path to the new license file
-------------------------	------------------------------

#### Returns

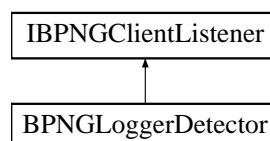
0 on failure, 1 on success

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

- [IBPNGClient.h](#)

## 7.9 IBPNGClientListener Struct Reference

Inheritance diagram for IBPNGClientListener:





## Public Member Functions

- virtual void WINAPI [onBPNGDeviceDetected](#) ([OnlineLoggerInfo](#) \*info)=0  
*Called to notify a detected logger in network.*
- virtual void WINAPI [onBPNGDeviceDisappeared](#) ([OnlineLoggerInfo](#) \*info)=0  
*Called to notify a disappeared logger.*
- virtual void WINAPI [onBPNGDeviceStateChange](#) ([OnlineLoggerInfo](#) \*info)=0  
*Called to notify a logger's state change.*
- virtual int WINAPI [onProgressDataDownload](#) (int percentCompleted)=0  
*Called to indicate the current progress of a file transfer.*
- virtual int WINAPI [onProgressConversion](#) (int percentCompleted, const char \*status)=0  
*Called to indicate the current progress of file conversion.*
- virtual void WINAPI [onStatusMessage](#) (const char \*statusMsg)=0  
*Called to send additional information of the current process to the calling app.*
- virtual void WINAPI [onWarning](#) ([BPNGWarningCode](#) warningCode, const char \*warnMsg)=0  
  
*Called to inform about a warning.*
- virtual int WINAPI [onTargetPathTooLong](#) (char \*newTarget, int maxSize)=0  
*Called on a too long target directory.*
- virtual int WINAPI [getOverwritingPermission](#) (const char \*filePath)=0  
*Called on existing output trace files.*
- virtual int WINAPI [onCriticalDiskSpace](#) (uint64\_t freeSpace, uint64\_t neededSpace, const char \*drive, const char \*msg)=0  
*Called in case of not enough free disk space.*
- virtual void WINAPI [onFirmwareUpdateProgress](#) (int percentage, int stepId, int subStepId, const char \*desc)=0  
*Called on firmware update progress.*
- virtual void WINAPI [onFirmwareUpdateError](#) (int errorId)=0
- virtual int WINAPI [onGetLogReportProgress](#) (int percentage, const char \*desc)=0

### 7.9.1 Member Function Documentation

**7.9.1.1** virtual int WINAPI IBPNGClientListener::getOverwritingPermission ( const char \* filePath ) [pure virtual]

Called on existing output trace files.

When an output trace file already exists this function is called. The listener has the possibility to return one of following values: -1: no, don't overwrite file -2: no, overwrite neither this nor any following file 1: yes, overwrite file 2: yes, overwrite this and all following files 0: cancel conversion

Implemented in [BPNGLoggerDetector](#).

**7.9.1.2** virtual void WINAPI IBPNGClientListener::onBPNGDeviceDetected ( [OnlineLoggerInfo](#) \* info ) [pure virtual]

Called to notify a detected logger in network.

All char\* of the passed [OnlineLoggerInfo](#)\* are only valid for the time of the function call. Please ensure to copy the string values.

Implemented in [BPNGLoggerDetector](#).

**7.9.1.3** virtual void WINAPI IBPNGClientListener::onBPNGDeviceDisappeared (   
OnlineLoggerInfo \* *info* ) [pure virtual]

Called to notify a disappeared logger.

All char\* of the passed OnlineLoggerInfo\* are only valid for the time of the function call. Please ensure to copy the string values.

Implemented in [BPNGLoggerDetector](#).

**7.9.1.4** virtual void WINAPI IBPNGClientListener::onBPNGDeviceStateChange (   
OnlineLoggerInfo \* *info* ) [pure virtual]

Called to notify a logger's state change.

All char\* of the passed OnlineLoggerInfo\* are only valid for the time of the function call. Please ensure to copy the string values.

Implemented in [BPNGLoggerDetector](#).

**7.9.1.5** virtual int WINAPI IBPNGClientListener::onCriticalDiskSpace ( uint64\_t *freeSpace*,   
uint64\_t *neededSpace*, const char \* *drive*, const char \* *msg* ) [pure virtual]

Called in case of not enough free disk space.

This notifies the listener about not enough free disk space for data download or conversion. The user can continue or abort the process. Returning 0 will abort the process. In some cases continuing without providing more disk space will call this function immediately again.

#### Parameters

<i>freeSpace</i>	Amount of free space
<i>neededSpace</i>	Amount of needed space
<i>drive</i>	Name of the drive where to store data
<i>msg</i>	Additional message to display

#### Returns

return 0 when process should be aborted, 1 to ignore

Implemented in [BPNGLoggerDetector](#).

**7.9.1.6** virtual int WINAPI IBPNGClientListener::onGetLogReportProgress ( int *percentage*,   
const char \* *desc* ) [pure virtual]

Called on creation of log report

#### Returns

return value 0 indicates an abort request from the implementing class

Implemented in [BPNGLoggerDetector](#).

**7.9.1.7** virtual int WINAPI IBPNGClientListener::onProgressConversion ( int *percentCompleted*, const char \* *status* ) [pure 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.

#### Parameters

<i>percent-Completed</i>	Percent of the entire conversion process (from 0...100%), -1 indicates the same value as from last function call
<i>status</i>	Status of the conversion process (e.g. "Converting trace data. Block 5 of 32")

#### Returns

return value 0 indicates an abort request from the implementing class

Implemented in [BPNGLoggerDetector](#).

**7.9.1.8** virtual int WINAPI IBPNGClientListener::onProgressDataDownload ( int *percentCompleted* ) [pure 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.

#### Parameters

<i>percent-Completed</i>	Percentage of the entire download process (from 0...100%)
--------------------------	---

#### Returns

return value 0 indicates an abort request from the implementing class

Implemented in [BPNGLoggerDetector](#).

**7.9.1.9** virtual void WINAPI IBPNGClientListener::onStatusMessage ( const char \* *statusMsg* ) [pure virtual]

Called to send additional information of the current process to the calling app.

This function transmit message strings to the listener class. Those messages are only for information purpose. The receiver doesn't have to react on it but can display it on the screen.

Implemented in [BPNGLoggerDetector](#).

**7.9.1.10** virtual int WINAPI IBPNGClientListener::onTargetPathTooLong ( char \* *newTarget*, int *maxSize* ) [pure virtual]

Called on a too long target directory.

Called when the resulting file name of the converted files is longer than the maximum allowed size of the file system (Windows 260). The lib user has to pass a new (shorter) base target directory to the passed char array with strcpy. The memory of the array is already allocated by the library and its size is maxSize. When a new directory was set the value 1 must be returned. If a listener wants to ignore this function call 0 has to be returned.

Implemented in [BPNGLoggerDetector](#).

**7.9.1.11** virtual void WINAPI IBPNGClientListener::onWarning ( BPNGWarningCode  
warningCode, const char \* warnMsg ) [pure virtual]

Called to inform about a warning.

This function transmit a warning message to the listener class. Warnings have a WARNING\_CODE and a warning message. Warnings do not interrupt the current process but should be noticed from the user to possibly initiate further provisions.

Implemented in [BPNGLoggerDetector](#).

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

- [IBPNGClientListener.h](#)

## 7.10 IChannel Struct Reference

Channel interface.

```
#include <BPNGDefines.h>
```

### Public Member Functions

- virtual [ChannelType](#) [getType](#) () const =0  
*Returns the ChannelType.*
- virtual uint8\_t [getIndex](#) () const =0  
*Returns the channel's index.*
- virtual const char \* [getName](#) () const =0  
*Returns the channel's name.*

### 7.10.1 Detailed Description

Channel interface.

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

- [BPNGDefines.h](#)

## 7.11 IChannelList Struct Reference

Channel list interface.

```
#include <BPNGDefines.h>
```

### Public Member Functions

- virtual uint8\_t [getSize](#) () const =0  
*Returns the number of channels.*
- virtual const [IChannel](#) \* [getChannel](#) (uint8\_t index) const =0  
*Returns the [IChannel](#) at index.*

#### 7.11.1 Detailed Description

Channel list interface.

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

- [BPNGDefines.h](#)

## 7.12 IConversionSet Struct Reference

A conversion set stores all conversion relevant settings.

```
#include <BPNGDefines.h>
```

### Public Member Functions

- virtual void [addChannel](#) ([ChannelType](#) channelType, uint8\_t channelIndex, [FormatId](#) formatId, int fileId=-1)=0  
*Adds a channel to the conversion set and assigns the target format to it.*
- virtual void [addTimeSpan](#) (uint64\_t startTime, uint64\_t endTime)=0  
*Adds a time span to the conversion set.*
- virtual void [addRdbldRange](#) (uint64\_t startId, uint64\_t endId)=0  
*Adds a ReferenceDB ID range to the conversion set.*

#### 7.12.1 Detailed Description

A conversion set stores all conversion relevant settings.

To convert trace data a conversion set must be created. Several channels can be added to one conversion set. The trace data of that channels are converted to the assigned formats. The conversion set also includes the data spans that has to be converted.

#### 7.12.2 Member Function Documentation

**7.12.2.1** virtual void [IConversionSet::addChannel](#) ( [ChannelType](#) *channelType*, uint8\_t *channelIndex*, [FormatId](#) *formatId*, int *fileId* = -1 ) [pure virtual]

Adds a channel to the conversion set and assigns the target format to it.

Use the `IBPNGClient::getLoggerChannel()` function to get all existing channels.

#### Parameters

<i>channelType</i>	must be one of the appropriate <code>ChannelType</code> enum.
<i>channelIndex</i>	zero-based channel index
<i>formatId</i>	must be one of the appropriate <code>FormatId</code> enum.
<i>fileId</i>	The data of all channels with same <code>formatId</code> and same <code>fileId</code> are written to the same output file. The default value -1 indicates always a separate file for each channel.

#### 7.12.2.2 `virtual void IConversionSet::addRdbIdRange ( uint64_t startId, uint64_t endId )` [pure virtual]

Adds a ReferenceDB ID range to the conversion set.

Passed parameter are IDs from the Reference Data Base (RDB). After calling on of the init functions `IBPNGClient::initOnline()` or `IBPNGClient::initOffline()` you can get the path to the RDB with `IBPNGClient::getReferenceDataBasePath()`.

The RDB includes all occurred events like startups, shutdowns, etc. but also all recorded trace files. Each RDB entry has a unique `DataBaseEntryId`. With this function you can easily select data between arbitrary RDB entries. For example you can convert all data between index X (which is e.g. a startup) and index Y (which is e.g. a shutdown). When the `DataBaseEntryId` of a trace file is passed, this trace block will be included by the conversion.

#### Parameters

<i>startId</i>	<code>DataBaseEntryId</code> that indicates the start of the data range to be converted
<i>endId</i>	<code>DataBaseEntryId</code> that indicates the end of the data range to be converted

#### 7.12.2.3 `virtual void IConversionSet::addTimeSpan ( uint64_t startTime, uint64_t endTime )` [pure virtual]

Adds a time span to the conversion set.

The data within the time span will be converted to the specified formats.

#### Parameters

<i>startTime</i>	must be in usec since 01.01.1970 (UTC)
<i>endTime</i>	must be in usec since 01.01.1970 (UTC)

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

- [BPNGDefines.h](#)

## 7.13 IFormatInfo Struct Reference

FormatInfo interface.

```
#include <BPNGDefines.h>
```

### Public Member Functions

- virtual [FormatId](#) [getFormatId](#) () const =0  
*Returns the FormatId.*
- virtual const char \* [getName](#) () const =0  
*Returns the format's name.*
- virtual BOOL [isMultipleChannelSupport](#) () const =0  
*Returns whether the format supports multiple channels in one output file.*
- virtual BOOL [isBinaryFormat](#) () const =0  
*Returns whether the format is binary.*
- virtual const char \* [getExtension](#) () const =0  
*Returns the format's default extension.*
- virtual int [getNumSupportedChannelTypes](#) () const =0  
*Returns the number of supported channel types.*
- virtual [ChannelType](#) [getChannelType](#) (int index) const =0  
*Returns one supported ChannelType.*

#### 7.13.1 Detailed Description

FormatInfo interface.

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

- [BPNGDefines.h](#)

## 7.14 IFormatList Struct Reference

Format list interface.

```
#include <BPNGDefines.h>
```

### Public Member Functions

- virtual int [getSize](#) () const =0  
*Returns the number of available formats.*
- virtual const [IFormatInfo](#) \* [getFormatInfo](#) (int index) const =0  
*Returns the IFormat at index.*

#### 7.14.1 Detailed Description

Format list interface.

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

- [BPNGDefines.h](#)

## 7.15 IRdbEvent Struct Reference

Interface to an RDB event.

```
#include <BPNGDefines.h>
```

### Public Member Functions

- virtual RdbEventType WINAPI [getType](#) ()=0  
*Get type of event.*
- virtual uint64\_t WINAPI [getUniqueld](#) ()=0  
*Returns the unique entry ID that can be set to DataSpans for data download and conversion.*
- virtual uint64\_t WINAPI [getTimeStamp](#) ()=0  
*Returns the event's time stamp in usec since 01.01.1970 UTC.*
- virtual const char \*WINAPI [getTimeZone](#) ()=0  
*Returns the logger's time zone that was active at the event's time stamp.*
- virtual int WINAPI [getIndex](#) ()=0  
*Returns the index of this event. Only used for marker events.*
- virtual const char \*WINAPI [getComment](#) ()=0  
*Returns additional information. The meaning of this string depends on the event's type. See RDB specification document for more information.*

### 7.15.1 Detailed Description

Interface to an RDB event.

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

- [BPNGDefines.h](#)

## 7.16 IRdbEventList Struct Reference

Interface to a list of rdb events.

```
#include <BPNGDefines.h>
```

### Public Member Functions

- virtual int WINAPI [getSize](#) ()=0  
*Returns the size of the event list.*
- virtual [IRdbEvent](#) \*WINAPI [getEvent](#) (int index)=0  
*Returns a pointer to the [IRdbEvent](#) at index.*



### 7.16.1 Detailed Description

Interface to a list of rdb events.

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

- [BPNGDefines.h](#)

## 7.17 MOSTPseudoMessagesProperties Struct Reference

MOST pseudo messages for each trigger can be written to MOST formats.

```
#include <BPNGDefines.h>
```

### Public Attributes

- **BOOL** [writeMessages](#)  
*Active flag for writing MOST pseudo messages for trigger.*
- **int** [source](#)  
*Source address.*
- **int** [target](#)  
*Target address.*
- **int** [functionBlockID](#)  
*Function block ID.*
- **int** [functionID](#)  
*Function ID.*
- **int** [reserved1](#)  
*reserved*
- **int** [reserved2](#)  
*reserved*
- **int** [reserved3](#)  
*reserved*
- **int** [reserved4](#)  
*reserved*

### 7.17.1 Detailed Description

MOST pseudo messages for each trigger can be written to MOST formats.

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

- [BPNGDefines.h](#)

## 7.18 OnlineLoggerInfo Struct Reference

Struct with information about a logger found in LAN.

```
#include <BPNGDefines.h>
```

## Public Attributes

- const char \* [ip](#)  
*the logger's ip address*
- const char \* [name](#)  
*the logger's name*
- const char \* [mbnr](#)  
*mainboard number*
- uint8\_t [connected](#)  
*connected with client, 0=false, else true*
- const char \* [currentUser](#)  
*user name of connected pc account*
- uint8\_t [loggerStatus](#)  
*current logger status,*

### 7.18.1 Detailed Description

Struct with information about a logger found in LAN.

### 7.18.2 Member Data Documentation

#### 7.18.2.1 uint8\_t OnlineLoggerInfo::loggerStatus

current logger status,

See also

[BPNGLoggerStatus](#)

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

- [BPNGDefines.h](#)

## 7.19 RdbEvent Struct Reference

Implementation class for a wrapper of [IRdbEvent](#) using STL classes.

```
#include <RdbEventList.hh>
```

## Public Member Functions

- **RdbEvent** ([IRdbEvent](#) \*rdbEvent)

## Public Attributes

- RdbEventType **type**
- uint64\_t **uniqueID**
- uint64\_t **timeStamp**
- std::string **timeZone**
- int **index**
- std::string **comment**

### 7.19.1 Detailed Description

Implementation class for a wrapper of [IRdbEvent](#) using STL classes.

To achieve a compiler independent interface for the blue PiraT 2 client library only pointer to complex objects are returned from some functions. The [IRdbEvent](#) class is can be wrapped by this class [RdbEvent](#) to have access to its members in the usual way. You only have to pass a [IRdbEvent](#) pointer to the constructor.

See also

[IRdbEvent](#), [RdbEventList](#)

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

- RdbEventList.hh

## 7.20 RdbEventList Class Reference

Implementation class for a wrapper of [IRdbEventList](#) using STL classes.

```
#include <RdbEventList.hh>
```

## Public Member Functions

- **RdbEventList** ([IRdbEventList](#) \*list)

### 7.20.1 Detailed Description

Implementation class for a wrapper of [IRdbEventList](#) using STL classes.

To achieve a compiler independent interface for the blue PiraT 2 client library only pointer to complex objects are returned from some functions. The class [IRdbEventList](#) is nothing else than a vector of [IRdbEvent](#) objects. Pass a pointer to [IRdbEventList](#) to the constructor of this wrapper class [RdbEventList](#) and you get a STL vector of [RdbEvent](#) objects which by itself is a wrapper to [IRdbEvent](#)

See also

[RdbEvent](#), [IRdbEventList](#), [IRdbEvent](#)

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

- RdbEventList.hh

## Kapitel 8

# File Documentation

### 8.1 BPNGDefines.h File Reference

Defines for blue PiraT 2 Library.

```
#include "stdint.h"
#include "eventID.h"
```

#### Classes

- struct [IChannel](#)  
*Channel interface.*
- struct [IChannelList](#)  
*Channel list interface.*
- struct [IFormatInfo](#)  
*FormatInfo interface.*
- struct [IFormatList](#)  
*Format list interface.*
- struct [IConversionSet](#)  
*A conversion set stores all conversion relevant settings.*
- struct [IRdbEvent](#)  
*Interface to an RDB event.*
- struct [IRdbEventList](#)  
*Interface to a list of rdb events.*
- struct [OnlineLoggerInfo](#)  
*Struct with information about a logger found in LAN.*
- struct [DataSpan](#)
- struct [BPNGError](#)  
*Error struct with error code and optional error message.*
- struct [CommonProperties](#)  
*Common properties.*
- struct [CANPseudoMessagesProperties](#)  
*CAN pseudo messages can be written to the CANoe ASC format.*

- struct [MOSTPseudoMessagesProperties](#)  
*MOST pseudo messages for each trigger can be written to MOST formats.*
- struct [ClientProperties](#)  
*The [ClientProperties](#) are a combination of [CommonProperties](#), [CanPseudoMessageProperties](#) and [MostPseudoMessageProperties](#).*

## Defines

- #define **WINAPI**
- #define **DECLDIR**
- #define **VOID** void

## Typedefs

- typedef eventid\_t **RdbEventType**
- typedef void(WINAPI \* [onLogRequest](#) )(const char \*logRecord)  
*Pointer to a function named onLogRequest with one parameter and no return value.*

## Enumerations

- enum [BPNGErrCode](#) {  
[BPNG\\_NOERR](#) = 0, [BPNG\\_LOGGER\\_NOT\\_FOUND](#) = 1, [BPNG\\_NOT\\_CONNECTED](#) = 2, [BPNG\\_CONNECT\\_FTP\\_FAILED](#) = 3,  
[BPNG\\_CONNECT\\_TMPBUS\\_FAILED](#) = 4, [BPNG\\_TMPBUS\\_NOT\\_CONNECTED](#) = 5, [BPNG\\_FTP\\_NOT\\_CONNECTED](#) = 6, [BPNG\\_FTP\\_SERVER\\_NOT\\_FOUND](#) = 7,  
[BPNG\\_FTP\\_LOGIN\\_FAILED](#) = 8, [BPNG\\_FTP\\_REMOTE\\_PATH\\_NOT\\_FOUND](#) = 9, [BPNG\\_FTP\\_READ\\_REMOTE\\_FILE\\_ERROR](#) = 10, [BPNG\\_FTP\\_WRITE\\_REMOTE\\_FILE\\_ERROR](#) = 11,  
[BPNG\\_FTP\\_TRANSFER\\_USER\\_CANCELED](#) = 12, [BPNG\\_FTP\\_CREATE\\_REMOTE\\_DIR\\_ERROR](#) = 13, [BPNG\\_FTP\\_REMOVE\\_REMOTE\\_DIR\\_ERROR](#) = 14, [BPNG\\_FTP\\_REMOVE\\_REMOTE\\_FILE\\_ERROR](#) = 15,  
[BPNG\\_FTP\\_CHANGE\\_CWD\\_ERROR](#) = 16, [BPNG\\_TMPBUS\\_COPYRDB\\_ERROR](#) = 17, [BPNG\\_TMPBUS\\_SEND\\_MSG\\_ERROR](#) = 18, [BPNG\\_TMPBUS\\_REQUEST\\_ERROR](#) = 19,  
[BPNG\\_FAILED\\_TO\\_CREATE\\_LOCAL\\_FILE\\_OR\\_DIRECTORY](#) = 20, [BPNG\\_LOCAL\\_PATH\\_NOT\\_FOUND](#) = 21, [BPNG\\_READ\\_LOCAL\\_FILE\\_ERROR](#) = 22, [BPNG\\_WRITE\\_LOCAL\\_FILE\\_ERROR](#) = 23,  
[BPNG\\_FILE\\_EXISTS\\_ERROR](#) = 24, [BPNG\\_DIR\\_EXISTS\\_ERROR](#) = 25, [BPNG\\_TARGET\\_PATH\\_TOO\\_LONG](#) = 26, [BPNG\\_ZIP\\_EXCEEDS\\_FATFS\\_MAX](#) = 27,  
[BPNG\\_XML\\_PARSER\\_ERROR](#) = 28, [BPNG\\_INITIALISATION\\_ERROR](#) = 29, [BPNG\\_RDB\\_SQLITE\\_QUERY\\_ERROR](#) = 30, [BPNG\\_RDB\\_OPEN\\_FAILED](#) = 31,  
[BPNG\\_CONVERSION\\_ERRORS](#) = 32, [BPNG\\_CONV\\_SET\\_NOT\\_FOUND](#) = 33, [BPNG\\_NOTHING\\_TO\\_CONVERT](#) = 34, [BPNG\\_TMT\\_FILE\\_ID\\_ERROR](#) = 35,  
[BPNG\\_TMT\\_FORMAT\\_ERROR\\_VERSION](#) = 36, [BPNG\\_TMT\\_FORMAT\\_ERROR\\_TS](#) = 37, [BPNG\\_INVALID\\_MESSAGE\\_ERROR](#) = 38, [BPNG\\_INVALID\\_MESSAGE\\_ID](#) = 39,  
[BPNG\\_INVALID\\_MESSAGE\\_TS](#) = 40, [BPNG\\_INVALID\\_MESSAGE\\_SUBID](#) = 41, [BPNG\\_INVALID\\_MESSAGE\\_LEN](#) = 42, [BPNG\\_CONV\\_FORMAT\\_ERROR](#) = 43,  
[BPNG\\_DOWNLOAD\\_ERRORS](#) = 44, [BPNG\\_NOTHING\\_TO\\_DOWNLOAD](#) = 45, [BPNG\\_INVALID\\_OFFLINE\\_SET](#) = 46, [BPNG\\_PARAMETER\\_MISMATCH](#) = 47,  
[BPNG\\_FW\\_VERSION\\_CHECK\\_ERROR](#) = 48, [BPNG\\_USER\\_CANCELLED](#) = 49, [BPNG-](#)

- `_MIN_VERSION_ERROR = 50, BPNG_EXCEPTION = 51,  
BPNG_INCOMPATIBLE_RDB = 52, BPNG_UNSPECIFIED_ERROR = 53, BPNG_CCP_  
XCP_PARSER_ERROR = 54, BPNG_CCP_XCP_DBC_GENERATOR_ERROR = 55,  
BPNG_CCP_XCP_SEQUENCE_GENERATOR_ERROR = 56, BPNG_INSUFFICIENT_  
DISK_SPACE = 57, BPNG_FWUPDATE_FAILED = 58 }`
- enum Error codes*
- `enum BPNGWarningCode { BPNG_NOWARNING, BPNG_WARNING_CLOSE_TRACE_  
FILES, BPNG_WARNING_MESSAGES_NOT_CONVERTED }`
- Warning codes.*
- `enum LanguageID { BPNG_GERMAN, BPNG_ENGLISH }`
- Languages.*
- `enum BPNGBugreportMode {  
BR_FULL_WO_TRACES = 0, BR_ONLY_LOGS = 1, BR_FDB_RDB = 2, BR_ONLY_CLI-  
ENT = 3,  
BR_FULL_ALL_TRACES = 4, BR_FULL_TIMESPAN_TRACES = 5 }`
- Mode for the IBPNG::downloadBugReport() function.*
- `enum ChannelType {  
CH_UNDEFINED = 0, CH_CANLS, CH_CANHS, CH_LIN,  
CH_SERIAL, CH_ETHERNET, CH_FLEXRAY, CH_MOST25_CTRL,  
CH_MOST25_MDP, CH_MOST25_SYNC, CH_MOST150_CTRL, CH_MOST150_MDP,  
CH_MOST150_MEP, CH_ANALOG_IN, CH_DIGITAL_IN, CH_CAMERA,  
CH_CCPXCP, CH_DIAG }`
- Currently supported interfaces.*
- `enum FormatId {  
UNDEFINED = 0, TMASC = 1, OP2 = 2, CANOE = 3,  
STA = 4, GNLOG = 5, TCLOG = 6, TCLOG_TS = 7,  
RAW_SERIAL = 8, RESERVED_1 = 9, IMG = 10, TMBIN = 11,  
CANCORDER = 12, APN = 13, ASCHEX = 14, WAV = 15,  
TCPDUMP = 16, BLF = 17, DLT_BMW = 18, MDF = 19,  
MDF_CAN_SIG = 20, MPEG4_BLOCKS = 21, MPEG4_JOINED = 22, SERIAL_DEBUG =  
23,  
RAW_ETHERNET = 24, INVALID = 0xFF }`
- Identifier for currently supported formats.*
- `enum BPNGLoggerStatus { LS_OK = 0, LS_ERROR = 1, LS_NOSYNC = 2, LS_UNDEFI-  
NED = -1 }`
- Logger status.*
- `enum DataSpanType { DST_IDSPAN = 0, DST_TIMESPAN = 1 }`
- Types for DataSpan.*

### 8.1.1 Detailed Description

Defines for blue PiraT 2 Library.

#### Author

Markus van Pinxteren

#### Date

12.05.2010

## 8.1.2 Enumeration Type Documentation

### 8.1.2.1 enum BPNGBugreportMode

Mode for the IBPNG::downloadBugReport() function.

Enumerator:

- BR\_FULL\_WO\_TRACES** Full bug report without traces.
- BR\_ONLY\_LOGS** Only log files are downloaded.
- BR\_FDB\_RDB** only FDB and RDB are downloaded
- BR\_ONLY\_CLIENT** only client logs are stored
- BR\_FULL\_ALL\_TRACES** Full bug report with all traces files.
- BR\_FULL\_TIMESPAN\_TRACES** Full bugreport with trace file of a specified time span.

### 8.1.2.2 enum BPNGErrCode

enum Error codes

An error is identified by one of the following error codes. Additional information may be found in the [BPNGError::msg](#) field (e.g. file path that causes a BPNG\_LOCAL\_PATH\_NOT\_FOUND error)

Enumerator:

- BPNG\_NOERR** no error
- BPNG\_LOGGER\_NOT\_FOUND** The IP address the lib wanted to connect was not found.
- BPNG\_NOT\_CONNECTED** A function call failed because the logger was not connected.
- BPNG\_CONNECT\_FTP\_FAILED** Establishing the ftp connection failed.
- BPNG\_CONNECT\_TMPBUS\_FAILED** Establishing the TMP (Telemotive Protocol) bus connection failed.
- BPNG\_TMPBUS\_NOT\_CONNECTED** TMP bus is not connected.
- BPNG\_FTP\_NOT\_CONNECTED** FTP is not connected.
- BPNG\_FTP\_SERVER\_NOT\_FOUND** FTP server is not found.
- BPNG\_FTP\_LOGIN\_FAILED** FTP login failed.
- BPNG\_FTP\_REMOTE\_PATH\_NOT\_FOUND** A requested path on the FTP server is not found.
- BPNG\_FTP\_READ\_REMOTE\_FILE\_ERROR** Can't read a file on the FTP server.
- BPNG\_FTP\_WRITE\_REMOTE\_FILE\_ERROR** Can't write a file on the FTP server.
- BPNG\_FTP\_TRANSFER\_USER\_CANCELED** FTP file transfer was canceled by the user.
- BPNG\_FTP\_CREATE\_REMOTE\_DIR\_ERROR** Can't create the directory on the FTP server.
- BPNG\_FTP\_REMOVE\_REMOTE\_DIR\_ERROR** Can't remove the directory on the FTP server.
- BPNG\_FTP\_REMOVE\_REMOTE\_FILE\_ERROR** Can't remove the file on the FTP server.
- BPNG\_FTP\_CHANGE\_CWD\_ERROR** Can't change the current working directory on the FTP server.



- BPNG\_TMPBUS\_COPYRDB\_ERROR** Failed to copy the reference data base to the logger's tmp directory.
- BPNG\_TMPBUS\_SEND\_MSG\_ERROR** Failed to send a TMP bus request message.
- BPNG\_TMPBUS\_REQUEST\_ERROR** The TMP bus request execution failed.
- BPNG\_FAILED\_TO\_CREATE\_LOCAL\_FILE\_OR\_DIRECTORY** Failed to create local file or directory.
- BPNG\_LOCAL\_PATH\_NOT\_FOUND** Local path not found.
- BPNG\_READ\_LOCAL\_FILE\_ERROR** Failed to read local file.
- BPNG\_WRITE\_LOCAL\_FILE\_ERROR** Failed to write local file.
- BPNG\_FILE\_EXISTS\_ERROR** Local file already exists.
- BPNG\_DIR\_EXISTS\_ERROR** Local directory already exists.
- BPNG\_TARGET\_PATH\_TOO\_LONG** Specified path exceeds the max. valid length (e.g. 260 for Windows systems)
- BPNG\_ZIP\_EXCEEDS\_FATFS\_MAX** ZIP file exceeds max size for FAT32 file systems.
- BPNG\_XML\_PARSER\_ERROR** Error while parsing xml file.
- BPNG\_INITIALISATION\_ERROR** BPNGClient instance is not initialised or with the wrong function. Use [IBPNGClient::initOnline](#) for data download or conversion directly from the device and [IBPNGclient::iniOffline](#) for data conversion from an offline data set.
- BPNG\_RDB\_SQLITE\_QUERY\_ERROR** Error when trying to read data from the rdb.
- BPNG\_RDB\_OPEN\_FAILED** Failed to open the reference data base.
- BPNG\_CONVERSION\_ERRORS** Multiple conversion errors. Use [IBPNGClient::getNumConversionErrors\(\)](#) and [IBPNGClient::getConversionError\(\)](#) for further information
- BPNG\_CONV\_SET\_NOT\_FOUND** The passed conversion set pointer was not created with this [IBPNGClient](#) instance and dus could not be found.
- BPNG\_NOTHING\_TO\_CONVERT** There is no data available that could be converted. Check the specified time/id spans.
- BPNG\_TMT\_FILE\_ID\_ERROR** Invalid TMT/XTMT file id while trying to convert data.
- BPNG\_TMT\_FORMAT\_ERROR\_VERSION** The TMT/XTMT version of the trace file is not supported by this lib version.
- BPNG\_TMT\_FORMAT\_ERROR\_TS** Missing FileTimeMessage in header of TMT/XTMT file.
- BPNG\_INVALID\_MESSAGE\_ERROR** Invalid messages found in trace file(s).
- BPNG\_INVALID\_MESSAGE\_ID** Invalid message id found in trace file(s).
- BPNG\_INVALID\_MESSAGE\_TS** Invalid message ts found in trace file(s).
- BPNG\_INVALID\_MESSAGE\_SUBID** Invalid message sub id found in trace file(s).
- BPNG\_INVALID\_MESSAGE\_LEN** Invalid message length found in trace file(s).
- BPNG\_CONV\_FORMAT\_ERROR** Invalid format assignment or mismatching recorded trace data for the specified conversion format.
- BPNG\_DOWNLOAD\_ERRORS** Multiple download errors. Use [IBPNGClient::getNumDownloadErrors\(\)](#) and [IBPNGClient::getDownloadError\(\)](#) for further information
- BPNG\_NOTHING\_TO\_DOWNLOAD** There is no data available that could be downloaded. Check the specified time/id spans.
- BPNG\_INVALID\_OFFLINE\_SET** Failed to initialise the [IBPNGClient](#) from the passed offline data set.

**BPNG\_PARAMETER\_MISMATCH** currently not used

**BPNG\_FW\_VERSION\_CHECK\_ERROR** The verification of the new firmware at the end of a firmware update failed.

**BPNG\_USER\_CANCELLED** currently not used

**BPNG\_MIN\_VERSION\_ERROR** The current library version does not suffice the the required min version written to [BPNGError::msg](#).

**BPNG\_EXCEPTION** Some kind of unhandled exception was thrown.

**BPNG\_INCOMPATIBLE\_RDB** The logger's or offline data set's RDB-Version is incompatible to this library version.

**BPNG\_UNSPECIFIED\_ERROR** An unspecified error occurred.

### 8.1.2.3 enum BPNGWarningCode

Warning codes.

Warnings are notified by listener calls to the function [IBPNGClientListener::onWarning\(\)](#)

Enumerator:

**BPNG\_WARNING\_CLOSE\_TRACE\_FILES** no warning Failed to close the current trace files on the logger device when trying to execute [IBPNGClient::initOnline\(\)](#)

**BPNG\_WARNING\_MESSAGES\_NOT\_CONVERTED** In case of protocol mismatch between recorded data and target format or unsupported message sub types, it is possible that some messages can not be converted to the selected format.

### 8.1.2.4 enum ChannelType

Currently supported interfaces.

Enumerator:

**CH\_UNDEFINED** undefined channel type

**CH\_CANLS** CAN low speed interface.

**CH\_CANHS** CAN high speed interface.

**CH\_LIN** LIN interface.

**CH\_SERIAL** Serial interface.

**CH\_ETHERNET** Ethernet interface.

**CH\_FLEXRAY** Flexray interface.

**CH\_MOST25\_CTRL** MOST 25 control channel.

**CH\_MOST25\_MDP** MOST 25 data packet channel (MDP)

**CH\_MOST25\_SYNC** MOST 25 synchronous channel (streaming data)

**CH\_MOST150\_CTRL** MOST 150 control channel.

**CH\_MOST150\_MDP** MOST 150 data packet channel (MDP)

**CH\_MOST150\_MEP** MOST 150 ethernet packet channel (MEP)

**CH\_ANALOG\_IN** Analog in.

**CH\_DIGITAL\_IN** Digital in.

**CH\_CAMERA** Camera channel.

**CH\_CCPXCP** CCP XCP.

### 8.1.2.5 enum FormatId

Identifier for currently supported formats.

Enumerator:

**TMASC** Telemotive ASCII format.  
**OP2** MOST OPformat.  
**CANOE** CANOE ASCII format.  
**STA** Serial Trace Analyser format.  
**GNLOG** GNLog format.  
**TCLOG** Trace Client format.  
**TCLOG\_TS** Trace Client format plus time stamps.  
**RAW\_SERIAL** Raw Serial Format.  
**RESERVED\_1** Reserved, formerly xml.  
**IMG** DataAnalyser IMG format.  
**TMBIN** Telemotive binary format.  
**CANCORDER** CANCorder format.  
**APN** APN serial format.  
**ASCHEX** ASCII format that writes binary data as hex values.  
**WAV** Wave PCM Format, use this for MOST Sync.  
**TCPDUMP** TCP dump (\*.pcap) format.  
**BLF** Binary Logging Format - Vector Informatik.  
**DLT\_BMW** BMW specific DLT Format.  
**MDF** Measurement data format - Vector Informatik.  
**MDF\_CAN\_SIG** MDF format based on CAN signals.  
**MPEG4\_BLOCKS** Camera format seperated blocks.  
**MPEG4\_JOINED** Camera joined videos.  
**SERIAL\_DEBUG** Serial Format with "new Line"-packaging.  
**RAW\_ETHERNET** Raw ethernet format.

### 8.1.2.6 enum LanguageID

Languages.

ID for specifying the language in that the library handles process and error information. Default language is english.

Enumerator:

**BPNG\_GERMAN** english  
**BPNG\_ENGLISH** german

## 8.2 IBPNGClient.h File Reference

Interface class for the BPNGClient DLL.

```
#include "BPNGDefines.h"
#include "IBPNGClientListener.h"
```

### Classes

- struct [IBPNGClient](#)  
*Interface class for the blue PiraT 2 client library.*

### Functions

- DECLDIR [IBPNGClient](#) \*WINAPI [getBPNGClient](#) (const char \*name="")  
*Factory function that creates instances of BPNGClient givin away ownership.*
- DECLDIR void WINAPI [setLanguageID](#) ([LanguageID](#) id)  
*Sets the language for status messages.*
- DECLDIR void WINAPI [addLogListener](#) ([onLogRequest](#) logFunc)  
*Adds a log listener to the library.*
- DECLDIR void WINAPI [removeLogListener](#) ([onLogRequest](#) logFunc)  
*Removes a log listener from the library.*

### 8.2.1 Detailed Description

Interface class for the BPNGClient DLL.

#### Author

Markus van Pinxteren

#### Date

21.04.2010

### 8.2.2 Function Documentation

#### 8.2.2.1 DECLDIR void WINAPI addLogListener ( [onLogRequest](#) *logFunc* )

Adds a log listener to the library.

The BPNG library writes debug traces to std::cout. If you want to process the debug outputs additionally (e.g. write to file) you can set a log listener to the lib. All set listeners get the log outputs from all BPNGClient instances.

All log outputs are forwarded to the registered listeners by calling the onLogRequest function that was added.

See also

[onLogRequest](#)

#### 8.2.2.2 DECLDIR IBPNGClient\* WINAPI getBPNGClient ( const char \* *name* = "" )

Factory function that creates instances of BPNGClient givin away ownership.

The instance is created on the heap and the allocated memory must be freed by the calling application. You can pass a name to this function. This will be the name of the created instance.

See also

[IBPNGClient::release\(\)](#), [IBPNGClient::getInstanceName\(\)](#)

## 8.3 IBPNGClientListener.h File Reference

Interface class for the BPNGClient listener.

```
#include "BPNGDefines.h"
```

### Classes

- struct [IBPNGClientListener](#)

#### 8.3.1 Detailed Description

Interface class for the BPNGClient listener.

##### Author

Markus van Pinxteren

##### Date

12.05.2010

# Index

- APN
  - BPNGDefines.h, [56](#)
- ASCHEX
  - BPNGDefines.h, [56](#)
- addChannel
  - IConversionSet, [42](#)
- addLogListener
  - IBPNGClient.h, [57](#)
- addRdbldRange
  - IConversionSet, [43](#)
- addTimeSpan
  - IConversionSet, [43](#)
- assignDBCFile
  - IBPNGClient, [28](#)
- BLF
  - BPNGDefines.h, [56](#)
- BPNG\_CONNECT\_FTP\_FAILED
  - BPNGDefines.h, [53](#)
- BPNG\_CONNECT\_TMPBUS\_FAILED
  - BPNGDefines.h, [53](#)
- BPNG\_CONV\_FORMAT\_ERROR
  - BPNGDefines.h, [54](#)
- BPNG\_CONV\_SET\_NOT\_FOUND
  - BPNGDefines.h, [54](#)
- BPNG\_CONVERSION\_ERRORS
  - BPNGDefines.h, [54](#)
- BPNG\_DIR\_EXISTS\_ERROR
  - BPNGDefines.h, [54](#)
- BPNG\_DOWNLOAD\_ERRORS
  - BPNGDefines.h, [54](#)
- BPNG\_ENGLISH
  - BPNGDefines.h, [56](#)
- BPNG\_EXCEPTION
  - BPNGDefines.h, [55](#)
- BPNG\_FAILED\_TO\_CREATE\_LOCAL\_FILE\_OR\_DIRECTORY
  - BPNGDefines.h, [54](#)
- BPNG\_FILE\_EXISTS\_ERROR
  - BPNGDefines.h, [54](#)
- BPNG\_FTP\_CHANGE\_CWD\_ERROR
  - BPNGDefines.h, [53](#)
- BPNG\_FTP\_CREATE\_REMOTE\_DIR\_ERROR
  - BPNGDefines.h, [53](#)
- BPNG\_FTP\_LOGIN\_FAILED
  - BPNGDefines.h, [53](#)
- BPNG\_FTP\_NOT\_CONNECTED
  - BPNGDefines.h, [53](#)
- BPNG\_FTP\_READ\_REMOTE\_FILE\_ERROR
  - BPNGDefines.h, [53](#)
- BPNG\_FTP\_REMOTE\_PATH\_NOT\_FOUND
  - BPNGDefines.h, [53](#)
- BPNG\_FTP\_REMOVE\_REMOTE\_DIR\_ERROR
  - BPNGDefines.h, [53](#)
- BPNG\_FTP\_REMOVE\_REMOTE\_FILE\_ERROR
  - BPNGDefines.h, [53](#)
- BPNG\_FTP\_SERVER\_NOT\_FOUND
  - BPNGDefines.h, [53](#)
- BPNG\_FTP\_TRANSFER\_USER\_CANCELED
  - BPNGDefines.h, [53](#)
- BPNG\_FTP\_WRITE\_REMOTE\_FILE\_ERROR
  - BPNGDefines.h, [53](#)
- BPNG\_FW\_VERSION\_CHECK\_ERROR
  - BPNGDefines.h, [55](#)
- BPNG\_GERMAN
  - BPNGDefines.h, [56](#)
- BPNG\_INCOMPATIBLE\_RDB
  - BPNGDefines.h, [55](#)
- BPNG\_INITIALISATION\_ERROR
  - BPNGDefines.h, [54](#)
- BPNG\_INVALID\_MESSAGE\_ERROR
  - BPNGDefines.h, [54](#)
- BPNG\_INVALID\_MESSAGE\_ID
  - BPNGDefines.h, [54](#)
- BPNG\_INVALID\_MESSAGE\_LEN
  - BPNGDefines.h, [54](#)
- BPNG\_INVALID\_MESSAGE\_SUBID
  - BPNGDefines.h, [54](#)
- BPNG\_INVALID\_MESSAGE\_TS
  - BPNGDefines.h, [54](#)
- BPNG\_INVALID\_OFFLINE\_SET
  - BPNGDefines.h, [54](#)
- BPNG\_LOCAL\_PATH\_NOT\_FOUND
  - BPNGDefines.h, [54](#)
- BPNG\_LOGGER\_NOT\_FOUND
  - BPNGDefines.h, [53](#)

- BPNG\_MIN\_VERSION\_ERROR
  - BPNGDefines.h, [55](#)
- BPNG\_NOERR
  - BPNGDefines.h, [53](#)
- BPNG\_NOT\_CONNECTED
  - BPNGDefines.h, [53](#)
- BPNG\_NOTHING\_TO\_CONVERT
  - BPNGDefines.h, [54](#)
- BPNG\_NOTHING\_TO\_DOWNLOAD
  - BPNGDefines.h, [54](#)
- BPNG\_PARAMETER\_MISMATCH
  - BPNGDefines.h, [54](#)
- BPNG\_RDB\_OPEN\_FAILED
  - BPNGDefines.h, [54](#)
- BPNG\_RDB\_SQLITE\_QUERY\_ERROR
  - BPNGDefines.h, [54](#)
- BPNG\_READ\_LOCAL\_FILE\_ERROR
  - BPNGDefines.h, [54](#)
- BPNG\_TARGET\_PATH\_TOO\_LONG
  - BPNGDefines.h, [54](#)
- BPNG\_TMPBUS\_COPYRDB\_ERROR
  - BPNGDefines.h, [53](#)
- BPNG\_TMPBUS\_NOT\_CONNECTED
  - BPNGDefines.h, [53](#)
- BPNG\_TMPBUS\_REQUEST\_ERROR
  - BPNGDefines.h, [54](#)
- BPNG\_TMPBUS\_SEND\_MSG\_ERROR
  - BPNGDefines.h, [54](#)
- BPNG\_TMT\_FILE\_ID\_ERROR
  - BPNGDefines.h, [54](#)
- BPNG\_TMT\_FORMAT\_ERROR\_TS
  - BPNGDefines.h, [54](#)
- BPNG\_TMT\_FORMAT\_ERROR\_VERSION
  - BPNGDefines.h, [54](#)
- BPNG\_UNSPECIFIED\_ERROR
  - BPNGDefines.h, [55](#)
- BPNG\_USER\_CANCELLED
  - BPNGDefines.h, [55](#)
- BPNG\_WARNING\_CLOSE\_TRACE\_FILES
  - BPNGDefines.h, [55](#)
- BPNG\_WARNING\_MESSAGES\_NOT\_CONVERTED
  - BPNGDefines.h, [55](#)
- BPNG\_WRITE\_LOCAL\_FILE\_ERROR
  - BPNGDefines.h, [54](#)
- BPNG\_XML\_PARSER\_ERROR
  - BPNGDefines.h, [54](#)
- BPNG\_ZIP\_EXCEEDS\_FATFS\_MAX
  - BPNGDefines.h, [54](#)
- BPNGDefines.h
  - APN, [56](#)
  - ASCHEX, [56](#)
  - BLF, [56](#)
  - BPNG\_CONNECT\_FTP\_FAILED, [53](#)
  - BPNG\_CONNECT\_TMPBUS\_FAILED, [53](#)
  - BPNG\_CONV\_FORMAT\_ERROR, [54](#)
  - BPNG\_CONV\_SET\_NOT\_FOUND, [54](#)
  - BPNG\_CONVERSION\_ERRORS, [54](#)
  - BPNG\_DIR\_EXISTS\_ERROR, [54](#)
  - BPNG\_DOWNLOAD\_ERRORS, [54](#)
  - BPNG\_ENGLISH, [56](#)
  - BPNG\_EXCEPTION, [55](#)
  - BPNG\_FAILED\_TO\_CREATE\_LOCAL\_FILE\_OR\_DIRECTORY, [54](#)
  - BPNG\_FILE\_EXISTS\_ERROR, [54](#)
  - BPNG\_FTP\_CHANGE\_CWD\_ERROR, [53](#)
  - BPNG\_FTP\_CREATE\_REMOTE\_DIR\_ERROR, [53](#)
  - BPNG\_FTP\_LOGIN\_FAILED, [53](#)
  - BPNG\_FTP\_NOT\_CONNECTED, [53](#)
  - BPNG\_FTP\_READ\_REMOTE\_FILE\_ERROR, [53](#)
  - BPNG\_FTP\_REMOTE\_PATH\_NOT\_FOUND, [53](#)
  - BPNG\_FTP\_REMOVE\_REMOTE\_DIR\_ERROR, [53](#)
  - BPNG\_FTP\_REMOVE\_REMOTE\_FILE\_ERROR, [53](#)
  - BPNG\_FTP\_SERVER\_NOT\_FOUND, [53](#)
  - BPNG\_FTP\_TRANSFER\_USER\_CANCELLED, [53](#)
  - BPNG\_FTP\_WRITE\_REMOTE\_FILE\_ERROR, [53](#)
  - BPNG\_FW\_VERSION\_CHECK\_ERROR, [55](#)
  - BPNG\_GERMAN, [56](#)
  - BPNG\_INCOMPATIBLE\_RDB, [55](#)
  - BPNG\_INITIALISATION\_ERROR, [54](#)
  - BPNG\_INVALID\_MESSAGE\_ERROR, [54](#)
  - BPNG\_INVALID\_MESSAGE\_ID, [54](#)
  - BPNG\_INVALID\_MESSAGE\_LEN, [54](#)
  - BPNG\_INVALID\_MESSAGE\_SUBID, [54](#)
  - BPNG\_INVALID\_MESSAGE\_TS, [54](#)
  - BPNG\_INVALID\_OFFLINE\_SET, [54](#)
  - BPNG\_LOCAL\_PATH\_NOT\_FOUND, [54](#)
  - BPNG\_LOGGER\_NOT\_FOUND, [53](#)
  - BPNG\_MIN\_VERSION\_ERROR, [55](#)
  - BPNG\_NOERR, [53](#)
  - BPNG\_NOT\_CONNECTED, [53](#)
  - BPNG\_NOTHING\_TO\_CONVERT, [54](#)
  - BPNG\_NOTHING\_TO\_DOWNLOAD, [54](#)
  - BPNG\_PARAMETER\_MISMATCH, [54](#)
  - BPNG\_RDB\_OPEN\_FAILED, [54](#)
  - BPNG\_RDB\_SQLITE\_QUERY\_ERROR, [54](#)
  - BPNG\_READ\_LOCAL\_FILE\_ERROR, [54](#)
  - BPNG\_TARGET\_PATH\_TOO\_LONG, [54](#)
  - BPNG\_TMPBUS\_COPYRDB\_ERROR, [53](#)
  - BPNG\_TMPBUS\_NOT\_CONNECTED, [53](#)
  - BPNG\_TMPBUS\_REQUEST\_ERROR, [54](#)

- BPNG\_TMPBUS\_SEND\_MSG\_ERROR, 54
- BPNG\_TMT\_FILE\_ID\_ERROR, 54
- BPNG\_TMT\_FORMAT\_ERROR\_TS, 54
- BPNG\_TMT\_FORMAT\_ERROR\_VERSION, 54
- BPNG\_UNSPECIFIED\_ERROR, 55
- BPNG\_USER\_CANCELLED, 55
- BPNG\_WARNING\_CLOSE\_TRACE\_FILES, 55
- BPNG\_WARNING\_MESSAGES\_NOT\_CONVERTED, 55
- BPNG\_WRITE\_LOCAL\_FILE\_ERROR, 54
- BPNG\_XML\_PARSER\_ERROR, 54
- BPNG\_ZIP\_EXCEEDS\_FATFS\_MAX, 54
- BR\_FDB\_RDB, 53
- BR\_FULL\_ALL\_TRACES, 53
- BR\_FULL\_TIMESPAN\_TRACES, 53
- BR\_FULL\_WO\_TRACES, 53
- BR\_ONLY\_CLIENT, 53
- BR\_ONLY\_LOGS, 53
- CANCORDER, 56
- CANOE, 56
- CH\_ANALOG\_IN, 55
- CH\_CAMERA, 55
- CH\_CANHS, 55
- CH\_CANLS, 55
- CH\_CCPXCP, 55
- CH\_DIGITAL\_IN, 55
- CH\_ETHERNET, 55
- CH\_FLEXRAY, 55
- CH\_LIN, 55
- CH\_MOST150\_CTRL, 55
- CH\_MOST150\_MDP, 55
- CH\_MOST150\_MEP, 55
- CH\_MOST25\_CTRL, 55
- CH\_MOST25\_MDP, 55
- CH\_MOST25\_SYNC, 55
- CH\_SERIAL, 55
- CH\_UNDEFINED, 55
- DLT\_BMW, 56
- GNLOG, 56
- IMG, 56
- MDF, 56
- MDF\_CAN\_SIG, 56
- MPEG4\_BLOCKS, 56
- MPEG4\_JOINED, 56
- OP2, 56
- RAW\_ETHERNET, 56
- RAW\_SERIAL, 56
- RESERVED\_1, 56
- SERIAL\_DEBUG, 56
- STA, 56
- TCLOG, 56
- TCLOG\_TS, 56
- TCPDUMP, 56
- TMASC, 56
- TMBIN, 56
- WAV, 56
- BR\_FDB\_RDB
  - BPNGDefines.h, 53
- BR\_FULL\_ALL\_TRACES
  - BPNGDefines.h, 53
- BR\_FULL\_TIMESPAN\_TRACES
  - BPNGDefines.h, 53
- BR\_FULL\_WO\_TRACES
  - BPNGDefines.h, 53
- BR\_ONLY\_CLIENT
  - BPNGDefines.h, 53
- BR\_ONLY\_LOGS
  - BPNGDefines.h, 53
- BP2Device, 17
- BPNGBugreportMode
  - BPNGDefines.h, 53
- BPNGDefines.h, 50
  - BPNGBugreportMode, 53
  - BPNGErrCode, 53
  - BPNGWarningCode, 55
  - ChannelType, 55
  - FormatId, 55
  - LanguageId, 56
- BPNGErrCode
  - BPNGDefines.h, 53
- BPNGError, 17
- BPNGLoggerDetector, 18
  - getOverwritingPermission, 19
  - onBPNGDeviceDetected, 19
  - onBPNGDeviceDisappeared, 19
  - onBPNGDeviceStateChange, 19
  - onCriticalDiskSpace, 19
  - onGetLogReportProgress, 20
  - onProgressConversion, 20
  - onProgressDataDownload, 20
  - onStatusMessage, 21
  - onTargetPathTooLong, 21
  - onWarning, 21
- BPNGWarningCode
  - BPNGDefines.h, 55
- CANCORDER
  - BPNGDefines.h, 56
- CANOE
  - BPNGDefines.h, 56
- CH\_ANALOG\_IN
  - BPNGDefines.h, 55
- CH\_CAMERA
  - BPNGDefines.h, 55
- CH\_CANHS
  - BPNGDefines.h, 55



- CH\_CANLS
  - BPNGDefines.h, [55](#)
- CH\_CCPXCP
  - BPNGDefines.h, [55](#)
- CH\_DIGITAL\_IN
  - BPNGDefines.h, [55](#)
- CH\_ETHERNET
  - BPNGDefines.h, [55](#)
- CH\_FLEXRAY
  - BPNGDefines.h, [55](#)
- CH\_LIN
  - BPNGDefines.h, [55](#)
- CH\_MOST150\_CTRL
  - BPNGDefines.h, [55](#)
- CH\_MOST150\_MDP
  - BPNGDefines.h, [55](#)
- CH\_MOST150\_MEP
  - BPNGDefines.h, [55](#)
- CH\_MOST25\_CTRL
  - BPNGDefines.h, [55](#)
- CH\_MOST25\_MDP
  - BPNGDefines.h, [55](#)
- CH\_MOST25\_SYNC
  - BPNGDefines.h, [55](#)
- CH\_SERIAL
  - BPNGDefines.h, [55](#)
- CH\_UNDEFINED
  - BPNGDefines.h, [55](#)
- CANPseudoMessagesProperties, [21](#)
- canPseudoMessages
  - ClientProperties, [23](#)
- ChannelType
  - BPNGDefines.h, [55](#)
- ClientProperties, [23](#)
  - canPseudoMessages, [23](#)
  - common, [23](#)
  - mostPseudoMessages, [23](#)
- common
  - ClientProperties, [23](#)
- CommonProperties, [24](#)
  - fileTimeSpansLikeSelection, [24](#)
  - separatedTimeFormat, [24](#)
  - separatedTimeFormatInOfflineSet, [25](#)
  - subfolderWithLoggerName, [25](#)
- connectLogger
  - Download Functions, [15](#)
- Conversion Functions, [16](#)
- convertData
  - IBPNGClient, [28](#)
- createCCPXCPFiles
  - IBPNGClient, [29](#)
- DLT\_BMW
  - BPNGDefines.h, [56](#)
- DataSpan, [25](#)
- deleteAllData
  - IBPNGClient, [29](#)
- deleteData
  - IBPNGClient, [29](#)
- Download Functions, [15](#)
  - connectLogger, [15](#)
- downloadBugReport
  - IBPNGClient, [29](#)
- downloadDataSpans
  - IBPNGClient, [30](#)
- fileTimeSpansLikeSelection
  - CommonProperties, [24](#)
- flashDeviceLED
  - IBPNGClient, [30](#)
- FormatId
  - BPNGDefines.h, [55](#)
- GNLOG
  - BPNGDefines.h, [56](#)
- getAvailableFormats
  - IBPNGClient, [31](#)
- getBPNGClient
  - IBPNGClient.h, [58](#)
- getConfig
  - IBPNGClient, [31](#)
- getConversionError
  - IBPNGClient, [31](#)
- getDeviceName
  - IBPNGClient, [31](#)
- getDownloadError
  - IBPNGClient, [32](#)
- getEventList
  - IBPNGClient, [32](#)
- getLastError
  - IBPNGClient, [32](#)
- getLoggerChannels
  - IBPNGClient, [32](#)
- getNumConversionErrors
  - IBPNGClient, [33](#)
- getNumDownloadErrors
  - IBPNGClient, [33](#)
- getOverwritingPermission
  - BPNGLoggerDetector, [19](#)
  - IBPNGClientListener, [38](#)
- getProperties
  - IBPNGClient, [33](#)
- getReferenceDataBasePath
  - IBPNGClient, [33](#)
- IMG
  - BPNGDefines.h, [56](#)
- IBPNGClient, [25](#)

- assignDBCFile, 28
- convertData, 28
- createCCPXCFiles, 29
- deleteAllData, 29
- deleteData, 29
- downloadBugReport, 29
- downloadDataSpans, 30
- flashDeviceLED, 30
- getAvailableFormats, 31
- getConfig, 31
- getConversionError, 31
- getDeviceName, 31
- getDownloadError, 32
- getEventList, 32
- getLastError, 32
- getLoggerChannels, 32
- getNumConversionErrors, 33
- getNumDownloadErrors, 33
- getProperties, 33
- getReferenceDataBasePath, 33
- initOffline, 34
- initOnline, 34
- keepLoggerAlive, 34
- reconfigLogger, 35
- release, 35
- scanNetworkForLogger, 35
- setDefaultConfig, 35
- setInfoEvent, 36
- setMarker, 36
- setProperties, 36
- setTime, 36
- updateFirmware, 36
- updateLicenses, 37
- IBPNGClient.h, 57
  - addLogListener, 57
  - getBPNGClient, 58
- IBPNGClientListener, 37
  - getOverwritingPermission, 38
  - onBPNGDeviceDetected, 38
  - onBPNGDeviceDisappeared, 38
  - onBPNGDeviceStateChange, 39
  - onCriticalDiskSpace, 39
  - onGetLogReportProgress, 39
  - onProgressConversion, 39
  - onProgressDataDownload, 40
  - onStatusMessage, 40
  - onTargetPathTooLong, 40
  - onWarning, 41
- IBPNGClientListener.h, 58
- IChannel, 41
- IChannelList, 41
- IConversionSet, 42
  - addChannel, 42
  - addRdbldRange, 43
  - addTimeSpan, 43
- IFormatInfo, 43
- IFormatList, 44
- IRdbEvent, 45
- IRdbEventList, 45
- initOffline
  - IBPNGClient, 34
- initOnline
  - IBPNGClient, 34
- keepLoggerAlive
  - IBPNGClient, 34
- LanguageID
  - BPNGDefines.h, 56
- loggerStatus
  - OnlineLoggerInfo, 47
- MDF
  - BPNGDefines.h, 56
- MDF\_CAN\_SIG
  - BPNGDefines.h, 56
- MPEG4\_BLOCKS
  - BPNGDefines.h, 56
- MPEG4\_JOINED
  - BPNGDefines.h, 56
- MOSTPpseudoMessagesProperties, 46
- mostPseudoMessages
  - ClientProperties, 23
- OP2
  - BPNGDefines.h, 56
- onBPNGDeviceDetected
  - BPNGLoggerDetector, 19
  - IBPNGClientListener, 38
- onBPNGDeviceDisappeared
  - BPNGLoggerDetector, 19
  - IBPNGClientListener, 38
- onBPNGDeviceStateChange
  - BPNGLoggerDetector, 19
  - IBPNGClientListener, 39
- onCriticalDiskSpace
  - BPNGLoggerDetector, 19
  - IBPNGClientListener, 39
- onGetLogReportProgress
  - BPNGLoggerDetector, 20
  - IBPNGClientListener, 39
- onProgressConversion
  - BPNGLoggerDetector, 20
  - IBPNGClientListener, 39
- onProgressDataDownload
  - BPNGLoggerDetector, 20
  - IBPNGClientListener, 40
- onStatusMessage

- BPNGLoggerDetector, [21](#)
- IBPNGClientListener, [40](#)
- onTargetPathTooLong
  - BPNGLoggerDetector, [21](#)
  - IBPNGClientListener, [40](#)
- onWarning
  - BPNGLoggerDetector, [21](#)
  - IBPNGClientListener, [41](#)
- OnlineLoggerInfo, [46](#)
  - loggerStatus, [47](#)
- RAW\_ETHERNET
  - BPNGDefines.h, [56](#)
- RAW\_SERIAL
  - BPNGDefines.h, [56](#)
- RESERVED\_1
  - BPNGDefines.h, [56](#)
- RdbEvent, [47](#)
- RdbEventList, [48](#)
- reconfigLogger
  - IBPNGClient, [35](#)
- release
  - IBPNGClient, [35](#)
- SERIAL\_DEBUG
  - BPNGDefines.h, [56](#)
- STA
  - BPNGDefines.h, [56](#)
- scanNetworkForLogger
  - IBPNGClient, [35](#)
- separatedTimeFormat
  - CommonProperties, [24](#)
- separatedTimeFormatInOfflineSet
  - CommonProperties, [25](#)
- setDefaultConfig
  - IBPNGClient, [35](#)
- setInfoEvent
  - IBPNGClient, [36](#)
- setMarker
  - IBPNGClient, [36](#)
- setProperties
  - IBPNGClient, [36](#)
- setTime
  - IBPNGClient, [36](#)
- subfolderWithLoggerName
  - CommonProperties, [25](#)
- TCLOG
  - BPNGDefines.h, [56](#)
- TCLOG\_TS
  - BPNGDefines.h, [56](#)
- TCPDUMP
  - BPNGDefines.h, [56](#)
- TMASC
  - BPNGDefines.h, [56](#)
- TMBIN
  - BPNGDefines.h, [56](#)
- updateFirmware
  - IBPNGClient, [36](#)
- updateLicenses
  - IBPNGClient, [37](#)
- WAV
  - BPNGDefines.h, [56](#)